Artificial Intelligence in Computer Science and Mathematics Education

David Azcona M.Sc.

Supervised by Prof. Alan F. Smeaton

Supervised by Prof. Sharon Hsiao (Arizona State University)

A thesis presented for the degree of Doctor of Philosophy (Ph.D.)

School of Computing

Dublin City University

August 2019

I hereby certify that this material, which I now submit for assessment on the programme of study leading to the award of Doctor of Philosophy is entirely my own work, that I have exercised reasonable care to ensure that the work is original, and does not to the best of my knowledge breach any law of copyright, and has not been taken from the work of others save and to the extent that such work has been cited and acknowledged within the text of my work.

David Azcona

David Azcona

ID No.: 15212605

August 2019

Acknowledgements

I would like to thank my tireless supervisor Prof. Alan F. Smeaton for his continuous support, patience, knowledge and encouragement over the last four years.

I would also like to thank my wonderful hosts at Arizona State University John Rome and Prof. I-Han Sharon Hsiao for believing in my potential and guiding me throughout the year where I conducted research as a Fulbright scholar in the United States.

I would also like express my deepest appreciation to my family and my girlfriend Maria for their daily encouragement. Thanks for being always there for me when I need it the most.

I am also indebted to Dr. Stephen Blott at the School of Computing in Dublin City University and the Action Lab at EdPlus in Arizona State University for their support and help.

Finally, I would like to acknowledge support from the following funding sources:

- Irish Research Council in partnership with The National Forum for the Enhancement of Teaching & Learning in Ireland under project number GOIPG/2015/3497
- Fulbright Ireland
- Science Foundation Ireland under grant number 12/RC/2289 (Insight Centre for Data Analytics)

- Dublin City University's School of Computing and Faculty of Engineering & Computing
- Arizona State University's University Technology Office and School of Computing, Informatics & Decision Systems Engineering
- Young European Research Universities Network

Contents

List of Figures

List of Tables

Abstract

Artificial Intelligence in Computer Science and Mathematics Education

David Azcona

In this thesis I examine how Artificial Intelligence (AI) techniques can help Computer Science students learn programming and mathematics skills more efficiently using algorithms and concepts such as Predictive Modelling, Machine Learning, Deep Learning, Representational Learning, Recommender Systems and Graph Theory.

For that, I use **Learning Analytics (LA)** and **Educational Data Mining (EDM)** principles. In Learning Analytics one collects and analyses data about students and their contexts for purposes of understanding and improving their learning and the environments students interact with. Educational Data Mining applies Data Mining, Machine Learning and statistics to data captured during these learning processes.

My central **research question** is how we can optimise the learning by students, of subjects like computer programming and mathematics in blended and online classrooms by mining and analysing data generated in these environments by the students. To validate the research question I have implemented several examples of monitoring student behaviour while learning, I have gathered various forms of student interaction data and combined it with demographics and student performance data (e.g. exam results) in order to test out different predictive models developed using a variety of AI and machine learning techniques. In these example environments

I have used these models not only to predict outcome and exam performance but also to automatically generate feedback to students in a variety of ways, including recommending better programming techniques. My research question is explored by examining the performance of the AI techniques in helping to improve student learning.

Chapter 1

Introduction

According to the 1st International Conference on Learning Analytics and Knowledge, which is an event where the topic really took off, became prominent and started becoming popular, Learning Analytics is the measurement, collection, analysis and reporting of data about learners and their contexts, for purposes of understanding and optimising learning and the environments in which it occurs. Higher education institutions across the world have been generating an enormous amount of raw data about students but they have traditionally been very inefficient in their use of this data and in putting it to good purpose [96]. With the data explosion in many fields such as Healthcare, Transportation and Business along with new methodologies now available and emerging to analyse and understand datasets, higher education institutions should become smarter organisations and should be moving towards using a data-first approach in their decision-making.

In addition to improving productivity, outputs and their own processes in educational institutions, there are strong employment opportunities for graduates in the Information and Communications Technology (ICT) sector with higher rates of employment and higher average salaries than most other graduates. In Ireland, according to the Higher Education Authority's 2019 report [85], computing courses have the highest level of student drop out, with close to half of all students in this area failing to complete their programme, across all third level institutions. Computer Science (CS) presents the lowest rate of completion at 55%, see Figure 1.1,

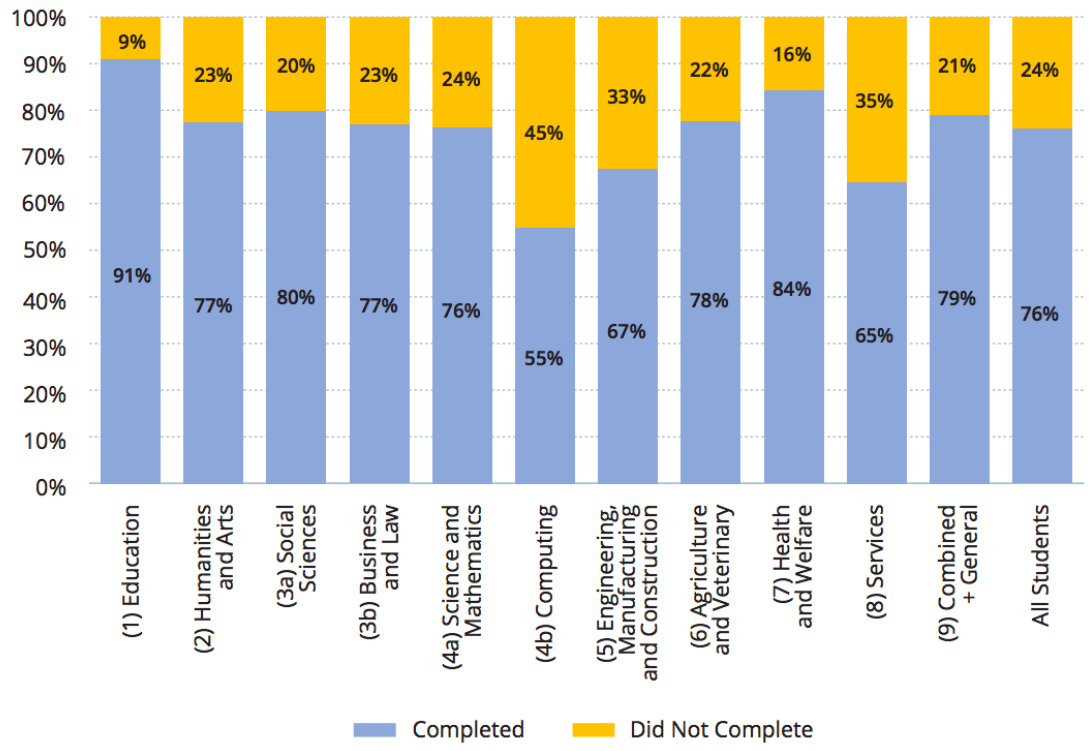


Figure 1.1: Higher Education Authority's Report on Overall Completion Rates by ISCED broad field of study

using the degree classification by the International Standard Classification of Education (ISCED). This comprehensive study tracked students who entered the third level system in 2007. A previous report showed the same trend [70].

A large contributor to these low progression rates in computing degrees is the fact that students quite often struggle on their first introductory programming course [102, 82]. The mean worldwide pass rate for introductory programming has been estimated at 67% [15], a figure revisited in 2014 [102].

Learning to program a computer is challenging for most people and few students find it easy at first. Moreover, first-year students often struggle with making the transition into University as they adapt to what is likely to be a very different form of independent study and learning. For instance, Dublin City University provides comprehensive orientation programmes and information for first-year students in order to ease this transition between second- and third-level education, as well as student support and career guidance for students at risk of stopping out (leave and

return) or dropping out (leave permanently) at any stage.

In this section of this introductory chapter, we will introduce the areas we will be discussing throughout this thesis, our motivation to work on them and the research questions we derived for each, which are used to frame the experiments carried out and reported later.

1.1 Introduction to Predictive Modelling

There are many different types of data which are gathered about our University students, ranging from static demographics to dynamic behaviour logs. These can be harnessed from a variety of data sources at Higher Education Institutions. Combining these into one location assembles a rich **student digital footprint**, which can enable institutions to better understand student behaviour and to better prepare for guiding students towards reaching their academic potential.

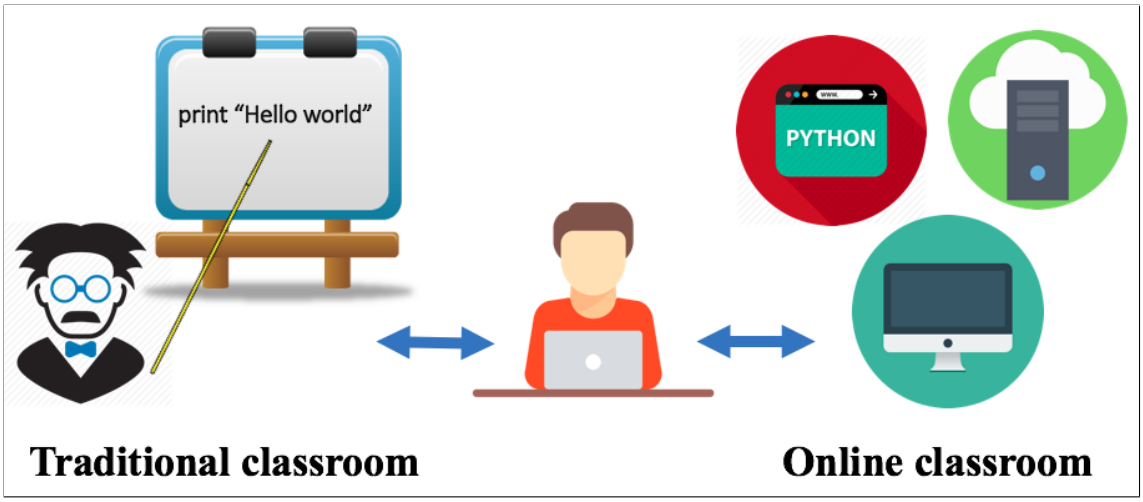


Figure 1.2: Blended Classrooms Combine Traditional Classrooms with Online Learning

In a recent literature review on learning computer programming, ten years of survey results highlighted that today's CS (Computer Sciences) classes still miss out on the use of diverse forms of Learning Analytics ([51]) to improve student per-

formance in the learning task, in some way. Automated collection of data on computer programming activities, the online activities that students carry out during their learning process, is typically used in isolation within designated programming learning environments such as WebCAT ([38]). Yet combining this automatically collected data with other complementary data sources (i.e. performance in class assignments or demographic information or information on prior learning) means it may have to be retrieved and aggregated from different course or University management systems. As a result, most of the data collection in the reported studies in CS learning is extremely customised and impossible to replicate and reproduce at other institutions.

Today, the majority of computer programming classes are delivered via a blended instructional strategy with face-to-face instruction in classrooms supported by online tools such as intelligent tutors, self-assessment quizzes, online assignment submission, and course management systems. New attempts in today's classrooms seek to combine multiple modalities of data such as gestures, gaze, speech or writing from video cameras, lecture recordings, etc. to leverage students' digital footprints [19, 79].

In this thesis, we propose, build and then evaluate a series of **traditional Machine Learning Predictive Analytics models** using student characteristics, prior academic history, students' programming laboratory work, and all logged interactions between students' offline and online resources. We generate predictions of end-of-course outcome weekly, during the semester. Furthermore, lecturers on the courses were updated each week regarding their students' progress.

This theoretical work then contributes to a practical implementation of a Predictive Analytics technique that aggregates multiple sources of students' digital footprints from blended classroom settings. This is done in order to validate the underlying theoretical work. This involves using multimodal data in the sense that it is derived from multiple sources of information about students and in the remainder of the thesis we refer to it as the students' digital footprint. Advanced Data Mining

techniques are adopted to develop and then create mathematical models to provide realtime prediction of course outcomes as well as personalised dynamic feedback to students on their progress. This approach incorporates static and dynamic student data features to enhance predictive model scalability that can be extrapolated to other blended classrooms and to other subjects as well as to other higher education institutions. Additionally, not only is the approach we develop generic, it also permits applicability in the case of only limited data sets being available (e.g. log files for access to online laboratory material only) in order to be beneficial in helping students in need. Most importantly, the generated predictions allow us to automatically create and provide adaptive feedback to each student according to each student's progression and also to provide guidance when in need.

We explore how the proposed Predictive Analytics models which are developed here, work in distinguishing students who may be struggling in computer programming courses. We have access to, and we use, two years of groundtruth student data as training data from which we can learn. To demonstrate the theory and address research questions, we implemented multimodal models for each course that aggregates sources of student data including student characteristics, prior academic history, students' programming laboratory work, and all the logged interactions between students' offline and online resources. Classification models are built by developing data features and automatically identifying and extracting patterns of success on these courses. These are then then trained and cross-validated to determine and then refine their accuracy, and finally predictions are generated every week with incoming student data. This gives us experimental data that we can use to validate our underlying research questions and hypotheses. A report containing whether each student is likely to pass or fail their next formal assessment and the associated confidence with that estimation, is sent to the lecturers for each course.

In summary, the single and most important research question derived in this section can be stated as the following:

RQ1: When working with new cohorts of University students about whom we have little historical interaction data, how accurate are the traditional Predictive Analytics models when used with generic static and dynamic student data features, in identifying those students in need of assistance in computer programming courses?

In the next section we will look at the technique of embeddings and how it can be used in this thesis.

1.2 Introduction to Representational Learning and Embeddings

Online learning tools and platforms including Massive Open Online Courses (MOOCs) provide a rich mechanism for students to engage and interact with educational material based on their individual existing knowledge and requirements for their own academic development. Such tools also provide a mechanism to support personalised learning effectively through the use of customised recommendations. These recommendations should be developed based on users' understanding, effort and their logged interaction with the learning systems to date by interpreting historical data from previous cohorts of students as well as data from the current students. Interest in, and the use of students' digital footprints and, particularly, interactions on VLE systems have been rising in the last decade because of their advantage in better supporting individualised learning. However, developing a **richer representation** for student digital footprints effectively and efficiently is still a challenging problem which has been an area of recent research interest, and is the focus of our work in this thesis.

Machine Learning (ML) is a subset of AI that provides computers with the ability to learn without being explicitly programmed [17]. That is done by combining the study of algorithms with statistical models. ML algorithms build a statistical model based on a collection of existing data with known outcomes such as retail data, bank loan applications or customer data from telecoms companies [57]. Those trained models are then used to predict outcomes for unknown data such as new

customers, new bank loan applications or new telecoms customers. There is a range of algorithms and ML techniques to do this including Support Vector Machines (SVM), Naive Bayes, Decision Trees and the recently popular Deep Learning or Neural Networks. New advances and techniques are being discovered regularly. One common challenge across all ML techniques and across all ML applications, is deciding what data to use to represent customers, bank loan applicants or telecoms customers, whatever the application is: which kind of customer data is most important, which is of little value and which can be discarded. This is sometimes called "data wrangling" and involves manual feature engineering including data cleaning and can take far more time than doing actual data analytics.

Representation Learning is a set of techniques in Machine Learning that allows a system to automatically discover the representations needed for feature detection or classification from raw data. This replaces manual feature engineering and allows a machine to both learn the features and use them to perform a specific task. For instance, learning richer distributed representations of words has shown to be quite effective for Natural Language Processing tasks [67, 13].

One of the main objectives of the work in this thesis in the area of learning analytics is to explore the latent signals or information buried in raw data by building high dimensional and distributional representations of student profiles and their programming codes or the outputs of their programming assignments. We propose a new methodology to profile individual CS students based on their programming design using a technique called embeddings. An embedding is a mapping from discrete objects to real number vectors. Such mappings constitute mapping to a dimension which may not always be meaningful or easily explainable in Machine Learning. However, the patterns of location and distances between vectors derived from embeddings may uncover numerous latent factors among the embeddings. In recent research in Deep Learning and Artificial Intelligence, the value of the amount of data has surpassed the complexity of the models. Thus, we investigate the use of hundreds of thousands of code submissions inputted to a **Deep Learning model**

using Embeddings as one of our implementation techniquess.

We will investigate different approaches to analyse student *source code* submissions in the Python language. We will compare the performances of different source code vectorisation techniques to predict the correctness of a code submission. In addition, we propose a new mechanism to represent students based on their code submissions for a given set of laboratory tasks on a particular course. Potentially, we can make deeper recommendations for programming solutions and pathways to support student learning and progression in computer programming modules effectively at a Higher Education Institution.

In further work we will investigate how to effectively represent and compare students' source code as submitted for assessment as part of their computer programming courses, on our internal online platform. We will investigate different techniques to represent *students' code* (code2vec) and evaluate the performance of different internal representations to predict the correctness of a code solution. Furthermore, after investigating different representations of user code (code2vec), we will propose a mechanism to represent students using their code submissions for given programming exercises for a course as a matrix (user2code2vec). We will investigate how this methodology can be used to effectively compare and rank students within in a class, to cluster students who show similar behaviour and to perform class-based analytics over the cohort of students.

The research questions that we investigate in this particular aspect of the thesis work can be enumerated as the following:

RQ2: How can students' programming submissions be encoded into vectors for use as internal representations of those students?

RQ3: By leveraging the vectorisation of code submissions for a given course, how can we represent students based on their programming work?

1.3 Introduction to Adaptive Feedback

As part of the experimental work in this thesis we conducted several semester-long classroom studies and collected data from several computer programming courses that adopted our approach to using predictive analytics in learning applications. In addition, during the second part of the semesters in which this was used, typically after students complete their first laboratory computer-based examination, students were free to opt-in to receive weekly personalised notifications. The feedback contained information regarding their predicted performance, based on the student data modalities gathered including their progress with laboratory sheets, programming code solutions, a form of peer feedback from predicted top-ranked students within the same class and university resources to reach out for help if needed, such as the University Student Support department, the course's lecturer or our system.

The feedback has been given via two methods:

- Weekly email notifications
- Virtual assistant (a WhatsApp ChatBot)

The accuracy of our Predictive Analytics models is crucial as students will receive customised feedback regarding their predicted performance. Then, we were able to measure the engagement with these customised notifications and how that could be an indicator of their performance. In addition, students were surveyed for their views and impressions.

The research questions derived from this subsection are the following:

RQ4: What are the effects of timely automatic adaptive support and peer-programming feedback on students' performance in computer programming courses?

RQ5: What are students' and teachers' perspectives and experiences after adopting a predictive modelling and adaptive feedback system into their own classes?

1.4 Introduction to Graph Theory and Networks

MOOCs are revolutionising education by giving students around the world open access to first-class education via the web. Lectures, readings, exercises and discussion forums are now one click away for anybody with an internet connection and a computer, anywhere. MOOCs gained popularity in 2012, which according to the New York Times, became "the year of the MOOC". Since then, the leading providers have been Coursera, Udacity and edX.

edX's mission is to increase access to high-quality education for everyone, everywhere and at the same time to enhance teaching and learning on campus and online. This MOOC provider, founded by Harvard University and MIT, is non-profit and open source. They offer courses from the world's best universities and institutions. On top of that, edX empowers research in education, pedagogy and learning by working with university partners. Their online environment is a great platform to explore how students learn and how lecturers can best teach their courses.

We will introduce a methodology to analyse large amount of students' learning states on two mathematics courses offered by the Global Freshman Academy program at Arizona State University. These two courses utilised ALEKS (Assessment and Learning in Knowledge Spaces) Artificial Intelligence technology to facilitate massive open online learning. We will explore network analysis and unsupervised learning approaches (such as probabilistic graphical models) on these types of Intelligent Tutoring Systems to examine the potential of the embedding representations that we develop in the thesis, on students' learning.

The single research question derived in this area is stated as the following:

RQ6: Can we extract valuable insights from massive open online learning platforms utilising the sequences of learning states?

1.5 Thesis Structure

This thesis consists of the following eight chapters:

(i) **Introduction**: the current chapter, which presents some of the context for the work reported later.

(ii) **Literature Review** (Chapter 2): this chapter explores state-of-the-art research in Learning Analytics and Educational Data Mining available in the literature.

(iii) **Students' Digital Footprints and Data Used in the Thesis** (Chapter 3): we introduce the datasets used for the studies which we use throughout the thesis, and these datasets are taken from two institutions.

(iv) **Modelling Student Online Behaviour** (Chapter 4): this chapter gives an overview of how to deploy a traditional Machine Learning model in an educational environment using students' digital footprints taken from that education environment.

(v) **Modelling Students With Embeddings** (Chapter 5): building on the previous chapter we explore how we can model students using their code submissions by leveraging the technique of embeddings.

(vi) **Adaptive Feedback to Students** (Chapter 6): we study how students improve their performance in end-of-semester module examinations based on the feedback provided to them.

(vii) **Using Graph Theory and Networks to Model Students** (Chapter 7): we look at graph theory to explore how students learn mathematical concepts.

(viii) **Conclusions** (Chapter 8): this final chapter summarises the research presented, revisits the research questions and asks have they been answered and proposes future directions for further research.

Chapter 2

Literature Review

2.1 Introduction

In this chapter we examine state-of-the-art literature regarding Artificial Intelligence techniques and how they may be used to help students to learn Computer Science programming skills and Mathematical concepts. These methods are broadly divided into Supervised Learning and Unsupervised Learning.

2.2 Machine Learning and Predictive Modelling

Research has shown that there has been significant interest in searching for the factors which motivate students to succeed in their first computer programming module as they master a programming skill set. In particular, researchers have been trying to identify the so-called "weak" students by looking at their characteristics, demographics, online and offline behaviour and performance in assessments [48]. Demographics, academic and psychological factors are all examples of static characteristics. When used for predicting computer programming success they include such things as prior programming knowledge [62], prior academic history like mathematics scores, number of hours playing video games and programming self-esteem [16, 88]. All these have been used in analysis of learning of computer programming, and with some success.

However useful these are, these factors do have some limitations [105]. First, this information has typically been gathered using written questionnaires. Lecturers in University settings have to process them and by the time they finish their course, some students may already have disengaged with their course. Second, and more importantly, these parameters do not reflect the students' actual effort and their learning progress throughout their course and might discourage students who are working on the material but possess characteristics like previous mathematics results, that are likely to present difficulties.

More recently, researchers have shifted their focus to a more data-driven approach to predicting student outcome by analysing computer programming behaviour, including patterns in compilations and programming states associated with the computer programs that students write and submit for assessment. These are substantially more effective at reflecting actual programming ability and competence, as well as progress in learning, than the characteristics on test performance gathered *prior* to the commencement of the course [104, 18].

The two main predictive measures are the Error Quotient [54] and the Watwin Score [105] which measure a student's behaviour between compilations and transitions in their learning from compilation errors. These metrics gather snapshots of the student's code on compilation using BlueJ or Microsoft Visual Studio with the OSBIDE plug-in while teaching using the Java or C++ programming languages and they potentially augment the programming environment to offer dynamic feedback or pathways.

Based on these predictors, new models, like the Normalised Programming State Model [23] which focuses on learning transitions, or data-driven approaches using machine learning, are emerging for these type of courses [1, 8]. In Computer Science education research, there are further studies to evaluate how students learn and to identify "at-risk" students by detecting changes in their behaviour as they learn computer programming, over time [39, 24].

In addition to using the computer programming behaviour of students as they

learn, students at Universities usually interact with an online Learning Management System (LMS) or Virtual Learning Environment (VLE) and in doing so they leave a digital trace or footprint. This has been leveraged previously to predict student performance in end-of-course exams across a range of subjects, not just computer programming. The most popular of the online educational systems are Moodle and Blackboard [25]. Moodle is an open source LMS while Blackboard (formerly WebCT) is a proprietary system.

Purdue University's Course Signals project predicts student performance using demographics, past academic history and learning effort as measured by interaction with their VLE, Blackboard. The predictive algorithm is run on demand by instructors and the outcome is fed back to students as a personalised email as well as a traffic signal which gives an indication of each student's prediction [7]. The Open University offers distance learning education and is the largest academic institution in the UK with more than 170,000 students. The OU Analyse project identifies and supports struggling students in more than 10 courses at different years of study. Lecturers may find it difficult to identify at-risk students without the feedback from face-to-face interactions that distance education has, but with predictive data at their finger tips they are able to identify, intervene, and support students and improve their virtual learning experience [60, 109]. Lastly, Dublin City University's Predictive Educational Analytics (PredictED) project used student interaction with the university's VLE, to predict likely performance of end-of-semester final grades for first year students across a range of topics. This project's interventions yielded nearly 5% improvement in absolute exam grade and proved that weekly automated feedback and personalised feedback to vulnerable first year students has a significant positive effect on their exam performance [28].

Learning Analytics have proven to provide a good indicator of how students are doing by looking at how online resources are being consumed. In computer programming classes and blended learning classrooms, students leave an even far greater digital footprint we can leverage to improve their experience and help to

identify those in need [51]. Combining learning analytics engagement features with programming states or behaviours in large classes can enable Lecturers to automatically identify students having difficulties at an earlier stage [58].

In research into Computer Science Education, based on the granularity, namely frequency and type of events, different models have been developed for student programming learning behaviour. The digital footprints used to drive these models include key strokes, program edits, compilations to executions and submissions [51]. In our work, explained in more detail in the following sections, we leverage an automated grading system for the teaching of programming. We collect submissions, a fine-grained footprint about each submission, and web logs regarding students' interactions with the material. However, we should note, we are limited by the frequency of the students submitting their solutions and we miss the programming actions in between.

2.2.1 Traditional Machine Learning in Practice

Machine learning algorithms build a mathematical model based on sample data (known as "training data") to make predictions without being explicitly programmed to perform a task [17]. We will now describe the steps followed in this thesis

(i) **Data management and storage**: data typically comes from files, databases or streams (for real time processing of live data). Files are good for distribution, and they can be structured or unstructured data. Databases are a good choice for centralised information and network access and the structure can be enforced using schemas. In our research, we sync all the students' programming file submissions to our own systems and we are also provided access to other files such as grades and student demographics. These files are usually in plain text format such as CSV or JSON which are human readable. Binary formats are also used for storing numeric arrays. In addition, when we develop web platforms for Faculty to look at we store this type of information in structured or unstructured databases.

(ii) **Data wrangling and cleaning**: datasets typically contain errors, inaccuracies, missing values, duplicates, inconsistencies, etc. Data wrangling is the process of transforming raw data into data we can process for extracting useful information. Raw data should be kept separate from cleaned data. Typically, we fix inaccuracies of the data and deal with missing values at this stage of our work.

(iii) **Data summarisation**: From the data distributions of each of the variables in our dataset, we analyse the measures of central tendency (statistics to capture the middle of the distributions) such as mean, median and mode as well as measures of statistical dispersion (statistics to measure how stretched each distribution is) such as variance, standard deviation and inter-quartile range. In addition, we can measure statistics of association between variables such as the covariance (how much two variables vary together), the linear (Pearson) and non-linear (Spearman) correlation coefficients (normalised version of the covariance for measuring the relationship between quantitative variables) and the mutual information (measure of the mutual dependence between two variables which is also knows as the "correlation for the 21st century" [99]). In our work, we confirm the predictive power of our features by analysing the correlation coefficients with a target variable. For instance, the programming percentage of work done by students is typically highly correlated with their performance on examinations.

(iv) **Data visualisation**: exploring the distributions of our variables and their relationships visually is incredibly useful. This provides us with sanity checks for our datasets and we can generate or confirm any hypothesis we may have at this stage. Visualisations are also key to communicate our hypotheses, conclusions and findings. For that, we typically use histograms for distributions, scatter plots for relationships between variables or bar charts to compare quantities.

(v) **Modelling**: in traditional Machine Learning a collection of features are handcrafted from the parameters or variables extracted from the data. This is a very manual process. In order to model the behaviour of, for instance students learning computer programming skills, we can approach it in two different ways:

- **Unsupervised Learning**: where we only have unlabelled data. In order to learn structure from the data, we assume the data forms distinct clusters (clustering) or data lies close to a lower dimensional manifold embedded in a high dimensional space (dimensionality reduction). Clustering techniques we can use are k-means, agglomerative clustering and Gaussian Mixture Models (GMM). The most popular dimensionality reduction techniques are PCA (Principal component analysis) and t-SNE (t-distributed stochastic neighbour embedding). In our work, as we will explain later, how we learned hidden structure from millions of transactions of students learning Mathematical concepts.
- **Supervised Learning**: here we have labelled examples, also known as groundtruth. There are two types of scenario:
  - **Regression**: when the output variable to be predicted is a real number. In our work, that would be equivalent to predicting an average grade of the year for the first-year university students in our university based on behavioural logs.
  - **Classification**: when the output variable to be predicted is a categorical variable. In our work, we predicted two categories, whether our students were "at-risk" or not.

  Our dataset will be split into three sets: training, validation and testing. Training will be used to fit the model and the validation data to optimise the hyperparameters of the learning function. Cross-Validation is a technique used to validate the model that repeatedly trains it and tests it on

a subset of the data (also known as folds). The testing set will be used to calculate the error using a scoring method such as Accuracy or F1-Score. Some of the more commonly used supervised learning algorithms are: Linear Regression (regression), Logistic Regression (classification), Decision Trees (typically classification), Random Forests (typically classification), Support Vector Machines (typically classification) and many more. A description of these different algorithms is beyond the scope of this thesis but can be found in any good online learning material or textbook such as [43] and [17].

2.3 Deep Learning and Embeddings

Deep Learning (DL) is an approach to designing and building AI systems and a subfield of of the broader area of machine learning. In ML, computers learn patterns from experience and past data, as was explained in more detail in the previous section. DL is a type of ML that represents or models the world as a nested hierarchy of concepts. More abstract concepts or representations are computed in terms of less abstract ones [43]. Figure 2.1 shows the relationship between AI disciplines and gives an example for each. Figure 2.2 is a high-level schematic of the relationship among some of these, with examples.

In the work in this thesis we are particularly interested in exploiting embeddings as an AI technique because of their applicability to modelling student behaviour, but first a little background into the topic. Neural language model-based distributed representations of text as proposed by [14] and further developed by [66, 67], learn distributed word representations using Neural Network based methods which are trained over large collections of text. These representations, commonly referred to as *embeddings*, embed an entire vocabulary into a comparatively low-dimensional vector space, where dimensions are real values. These embedding models have been shown to perform well on semantic similarity between words and on word analogies tasks [67, 13].

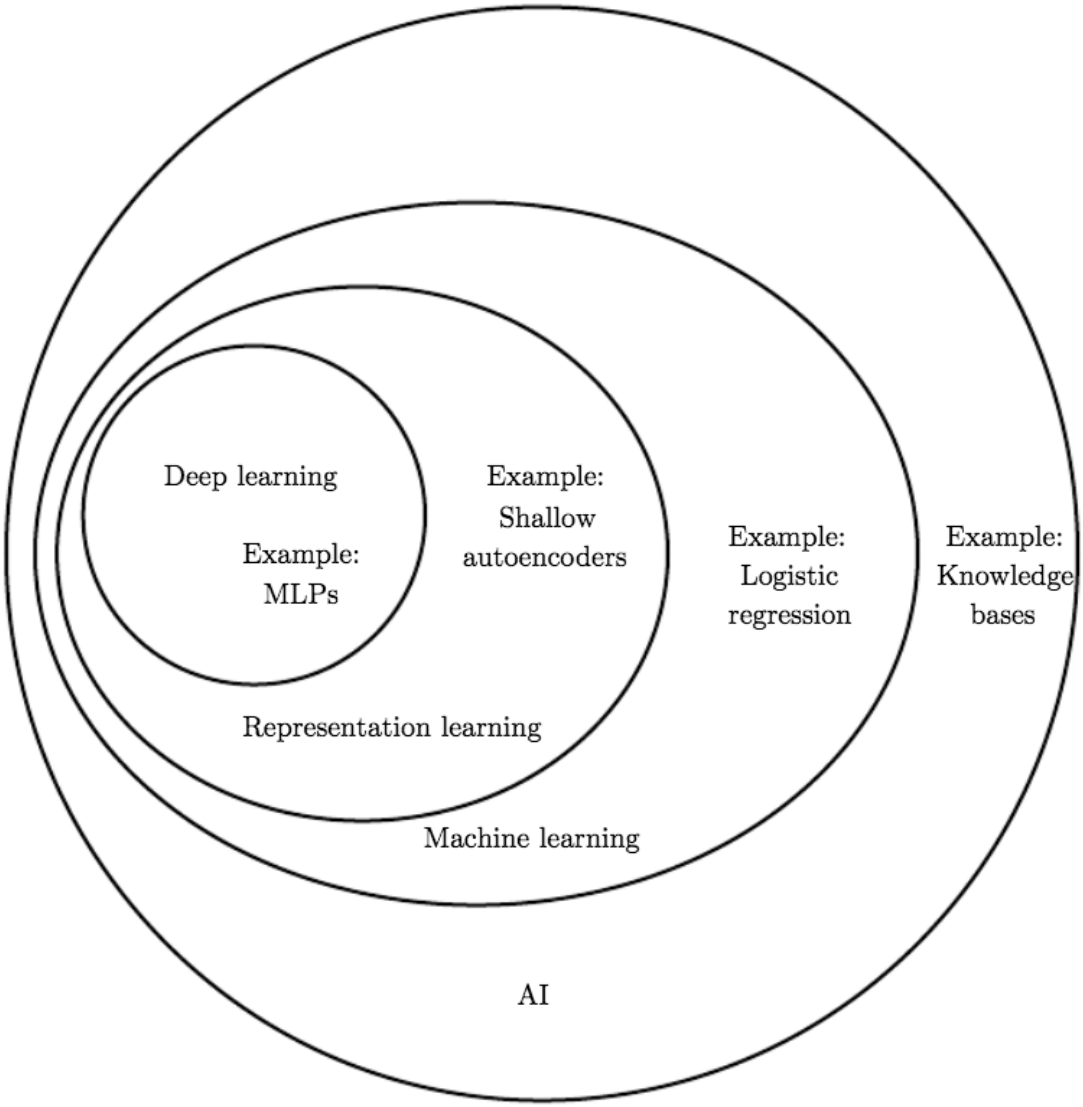


Figure 2.1: Venn diagram showing how deep learning is a kind of representation learning, which is in turn a kind of machine learning, which is used for many but not all approaches to AI. Image taken from [43].

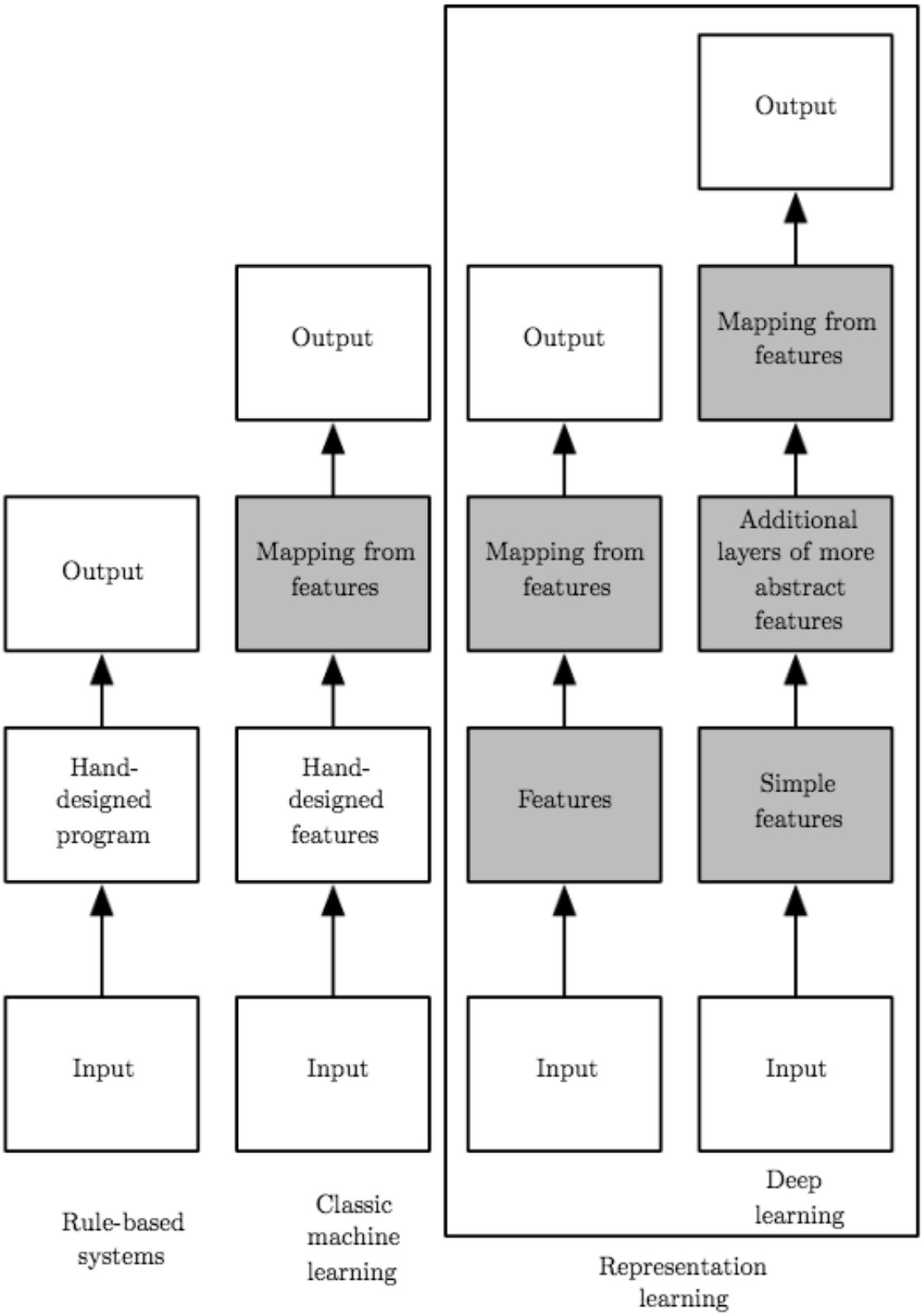


Figure 2.2: Flowcharts showing how the different parts of an AI system relate to each other within different AI disciplines. Shaded boxes indicate components that are able to learn from data. Image taken from [43].

In the area of computer programming, predicting code properties or extracting meaningful features from vast amounts of code data has experienced tremendous progress recently [4, 83, 6]. Predicting code properties without compiling or running is used for name prediction of program entities [5], code generation [90], code completion [91] and code summarisation [3]. In addition, embeddings-based techniques have been recently applied to learning effective code representations, comparing source codes and recommending approaches to students.

[71] recently proposed how to successfully develop program vector representations to be used in conjunction with Deep Learning models for the task of classifying computer programs. The vector representations learned used the nodes from Abstract Syntax Trees (ASTs) which are a tree representation of the abstract syntactic structure of source code [77]. The authors explored other granularity levels for representations such as characters, tokens or statements. In our work, we also explore tokens as a way to vectorise code submissions by leveraging the Python Tokeniser library.

Even more recently, [6] developed a code2vec neural attention network that collects AST paths and aggregates them to extract syntactic information from code snippets. Their objective was to predict semantic properties such as method names by representing snippets of code as continuous distributed vectors, also known as Code Embeddings. In our work, we build similar higher-level distributed vectors to predict the correctness of code solutions to verify patterns and meaningful information is then extracted.

[83] leveraged Code Embeddings to give feedback to students in MOOCs. First, they captured functional and stylistic elements of student submissions and, then, they learned how to give automatic feedback to students. This was done by developing functionality matrices at each point of the syntax tree of the submission.

In terms of providing student feedback, [80] demonstrated a continuous hint approach can predict what capable students would do in solving a multi-step programming task and that the hints built using embeddings can match the edit hints

that human tutors would have given. Also, [44] proposed feedback strategies and automatic example assignments using structured solution spaces. More recently, [86] collected a dataset of rich events streams. Instead of studying artifacts after they happened, they build FeedBaG, a general-purpose interaction tracker for Visual Studio that monitors development activities and collected data from software developers.

Finally, [72] proposed a tree-based Convolutional Neural Network, denoted as TBCNN, using a convolution kernel designed over programs' ASTs to capture structural information. They also used this technique to classify programs based on functionality and detecting code snippets with particular patterns. In addition, developing a dataset of syntax trees can be used for recommendations as [87] did for C# using solutions taken from GitHub.

In our work, code solutions from students are transformed into continuous distributed vectors, Code Embeddings, to be used as a representation of their programming submissions (code2vec). These vectors are leveraged to construct a matrix that represents each user in a comparable way (user2code2vec). [93] proposed a Tensor Factorisation approach for modelling learning and predicting student's performance that does not need any prior knowledge. This work outperformed state-of-the-art approaches for measuring learning and predicting performance such as Bayesian Knowledge Tracing and other tensor factorisation approaches. We were inspired by this work [93] to develop a similar representation for users who learn coding at our University and we use embeddings to learn higher level representations of that information.

Deep learning methods for machine learning, are representation-learning methods with multiple levels of representation, obtained by stacking multiple simple but non-linear layers with weights which are learned from data [61]. These culminate in an output which may be a category, or a number, or multiples. The way in which the deep learning architecture operates, we learn increasingly more abstract concepts. As the amount of data available to deep learning algorithms increases, accuracy of

the predicted output does as well, substantially outperforming traditional feature extraction techniques combined with traditional machine learning algorithms [78], across a range of applications.

We consider the following type of neural networks:

- **Feed-forward Neural Networks**: where each neuron in the input layer is connected to every output neuron in the next layer (fully connected layer).
- **Convolutional Neural Networks (CNNs)**: where each layer in a CNN applies a different set of filters (also known as convolutions) and combines the result before applying an activation function such as Rectified Linear Unit (ReLU). In image classification applications, these convolutional layers can learn edges and shapes and, on top of those layers, higher-level features such as facial structures.
- **Recurrent Neural Networks (RNNs)**: where the connections between nodes form a directed graph along a temporal sequence. This method uses its memory to process sequences of inputs.

In contrast to neural network architectures, mathematical embeddings involve converting an item from a space with many dimensions to a continuous vector space with a much lower number of dimensions. The most popular methods to generate this mapping are neural networks and dimensionality reduction. In our case, we leverage neural networks. By framing a problem as a prediction, for instance, and including an embeddings layer with a much lower dimension we force the network to learn higher level representations. This will be further explored in Chapter 5.

This section has presented a fairly high level overview of deep learning but there is much more that we could describe, including gradient descent, different kinds of activation functions, initialisation of the network, regularisation, and more, but these are outside the scope of this thesis where we want to use these techniques rather than develop new ones. The reader is referred to the well-cited book by Goodfellow *et al.*, for details on this topic [43].

2.4 Adaptive Feedback in Learning

Feedback is an effective way to motivate novice programmers and an important research avenue for teaching computer programming. Recently, researchers have been working on augmenting the IDE or programming environment by crowdsourcing code solutions. Students are suggested error corrections or solutions that their peers have applied before. Java has been the programming language initially targeted for this approachwith systems such as BlueFix [103] and HelpMeOut [46]. This latter social recommender system was also applied to learning on the Arduino platform. In addition, Crowd::Debug [73] was presented as a similar solution for the Ruby programming environment, a test-driven development language.

In terms of notifying students as to how they are progressing throughout the semester, Purdue University's Course Signals [7] sent a personalised mail and posted a traffic signal as an indication of their performance and Dublin City University's PredictED [28] project notified students how they were doing and where they were rated in terms of progress within their own class. Both systems yielded impressive improvement in first-year retention rates. In our work, our programming grading system provides real-time feedback on computer program submissions by running a suite of testcases but it does not provide code suggestions or solutions for errors that the programs may have.

Feedback has always been one of the most effective methods in enhancing students' learning [47]. There is an abundance of factors that affect educational achievement. Some factors are more influential than others. For instance, feedback types and formats and the timing of providing feedback [95] are both important. Studies have reported that positive feedback is not always positive for students' growth and achievement [47]; "critical" rather than "confirmatory" feedback is the most beneficial for learning regardless of whether feedback was chosen or assigned [31]. Content feedback achieves significantly better learning effects than progress feedback, where the former refers to qualitative information about the domain content and its accuracy, and the latter describes the quantitative assessment of the stu-

dent's advancement through the material being covered [53]. Several of the different feedback factors were explored at the intersection with the learner's variables (i.e. skills, affects) and reported to support personalised learning [75]. For instance, cognitive feedback was found to make a significant difference in the outcomes of student learning gains in an intelligent dialogue tutor [21]. Students' affects were adapted to improve motivational outcome (self-efficacy) in work reported in [21, 32] while using student characteristics as input to tutoring feedback strategies to optimise students' learning in adaptive educational systems was reported in [76]. While a large body of empirical studies investigate the feedback impacts in the context of learning [110], we focused on researching educational technology to support delivering adaptive feedback for computer programming courses.

Recommender systems have provided numerous novel applications in industry including recommendation of books, movies, music and more. Recommender systems can offer the education sector a new direction of research into technology enhanced learning [64] by including recommender technology in learning platforms. In the context of learning computer programming, several studies implemented resource recommendation or best next-item or step recommendation (i.e. the next best item to be viewed in the navigational sequence) [49], and these have reported promising approaches and results. For instance, CodeReco [97], CodeBroker [111] and SnipMatch [107] all use similarity measures to recommend partial source code to facilitate problem solving while OOPS recommends relevant worked examples [98]. In this work, we focus on recommending relevant resources based on learner's needs.

2.5 Graph Theory and Networks

In the final topic for our literature review, we briefly summarise work in the area of graph theory and networks. The reason for including this topic is that a system called ALEKS (Assessment and Learning in Knowledge Spaces), which we describe in the next Chapter in Section 3.2.2, leverages AI techniques in order to map students' knowledge. ALEKS is based on knowledge spaces, which was introduced in

1985 by Doignon and Falmagne, who describe the possible states of knowledge of a learner [35]. In order to develop a knowledge space, a domain like Algebra or Chemistry is modelled and divided into a set of concepts and feasible states of knowledge where the student's knowledge is at any given point in time. This technology adapts and navigates the students by determining what the student may know and may not know in a course and guides her to the topics she is most ready to learn. It assesses the student's knowledge periodically to ensure topics are learned and retained [36]. Recent research has shown that using ALEKS for learning Mathematics has a positive learning impact on an after-school program for more than 200 sixth graders [29, 30].

Related to our use of data from the ALEKS system, recent research has shown that Network Analysis measurements can be used as predictive features for machine learning models in addition to generic content-based features [26]. Moreover, sequential modelling (i.e. Hidden Markov Models (HMMs)) can be useful to uncover student progress or students' learning behaviours [84, 83, 50]. We hypothesise that modelling the evolution of a large number of students' working behaviours with social network features, will allow us to uncover students' progression. This, in turn, will allow the possibility to enhance the student experience with further personalised interventions in these Intelligent Tutoring Systems as they gather rich information about concepts, topics and learning states.

Chapter 3

Students' Digital Footprints and the Data Used in the Thesis

In the work reported in this thesis, we leveraged data from two Higher Education Institutions, namely Dublin City University (DCU) and Arizona State University (ASU).

Higher Education Institutions collect data about their students at multiple points during the student journey and they store this in different locations and on different institutional systems. This data includes information on students' background and demographics at the time of initial registration, interaction with the institution's online learning environments and other online resources like WiFi access, online library resources and student support services, geolocated data from physical locations like lecture attendance or library accesses, and some aspects of their social activities like memberships of clubs and societies. Leveraging all these sources of information and many more, if they were integrated together, could shape a picture of the students' engagement and involvement on campus.

Moreover, in learning discipline-specific subjects such as computer programming, students spend a considerable amount of their time in laboratory sessions. While learning computer programming, students typically interact with a platform to develop and submit their program code for specified assignments problems leaving an

even greater digital footprint than learning in other disciplines [51, 52]. Some of these platforms are: Web-CAT [38], CloudCoder [81], CodeWorkout [37], Blackbox [22] using the BlueJ plugin [59] and many more.

These types of computer program submission platforms are often used to evaluate the correctness of the students' computer programming work which acts as a measure of their progression through the course, and the effectiveness of their learning. Analytics platforms could be used to make use of this information in order to understand students' engagement and behaviour, which could, in turn, be an indicator of their learning experience. Unfortunately, these automated assessment systems are not the only tools that the students and the instructors will use, especially when students take multiple courses or modules in the computer science area, and they often have to switch among several online educational platforms for each course. Therefore, without collecting all the diverse interaction data, plus all the other data on students that an institution has, it is challenging to establish reliable groundtruth data in order to train predictive models.

In summary, at Dublin City University, we collected student data from Computer Science students learning how to code in programming modules and, in addition, from first-years at the whole university. In Arizona State University, we collected student data from reviewing behaviours in a programming course and from Mathematics MOOCs taught at the EdX platform. Table 3.1 shows a summary of the data used in the thesis.

Table 3.1: Summary of the Data Used in the Thesis

University	School	Type	Source
DCU	Computing	Programming	DCU's Grading Assistant
	All	Usage of Resources	Various
ASU	Computing	Programming	ASU's Grading Assistant
	Online	Learning States	EdX & ALEKS

3.1 Dublin City University

Dublin City University is a university based on the Northside of Dublin, Ireland. Created as the National Institute for Higher Education, Dublin in 1975, it enrolled its first students in 1980, and was elevated to university status in September 1989 by statute. The university has 17,000 students and around 1,200 online distance education students.

3.1.1 School of Computing

Dublin City University offers two honours Bachelor degrees in Computer Science through its School of Computing, a B.Sc. in Computer Applications (CA) and a B.Sc. in Enterprise Computing (EC). CA prepares students for a career in computing and information technology by giving them in-depth knowledge of software engineering and the practical skills to apply this knowledge to develop the technology behind computing-based products. EC prepares students to use computing technology to help organisations to work together and give companies a competitive edge in the marketplace. The EC degree is more focused on topics like managing information technology and developing and using systems to improve and even to re-design the way organisations do business. It is safe to say, CA teaches students deeper computer programming skills and EC is more business and project management oriented.

The data used in this thesis from DCU was drawn from students registered in the CA and EC degree programs. Specifically, the data sources we made use of in order to model student interaction, engagement and effort in computer programming courses in DCU consists of:

- **Student Characteristics**: gender, date of birth, citizenship and domicile.
- **Prior Academic history**: prior-to-university test scores: Irish CAO points and Leaving Certificate exam scores (equivalent to GPA and SAT exams in the US) and prior academic history at the university if there is any.

- **Interaction logs**: Students interact online with the custom VLE developed for computer programming courses and every instance of a student's access to a page of any kind is recorded and stored. These are web logs from an Apache web server for the resource or page requested, the date and time of access, the unique student identifier, and the IP address of the device used for access.

- **Programming submissions**: A custom Virtual Learning Environment (VLE) for the teaching of computer programming has been developed by Dr. Stephen Blott [1]. This automated grading platform is used in a variety of computer programming courses across the Faculty including students in both the CA and EC degree programmes. Using this VLE, students can browse course material (as on any LMS) and submit and verify their computer programming laboratory work. Figure 3.1 shows how students are able drag and drop their program files onto the platform. Figure 3.2 shows the real-time feedback students get when verifying a program by running a suite of pre-specified testcases on the grading server. The analytics information extracted for each submission is the program name, code, laboratory sheet that it belongs to, whether the submission is correct or incorrect according to the lecturer's testcases, and the date and time of the submission.

Table 3.2 presents a list of the courses we used to test our various research questions and hypothesis that will be discussed in the following sections. Some of these courses were delivered multiple times and we use data gathered from multiple runnings of these courses.

These courses use the custom VLE for the teaching of computer programming which allows us to capture a fine-grained digital footprint of students interacting with computer programming learning material and submitting their code solutions.

[1] Dr. Stephen Blott is an Associate Professor at the School of Computing in Dublin City University http://www.computing.dcu.ie/~sblott/

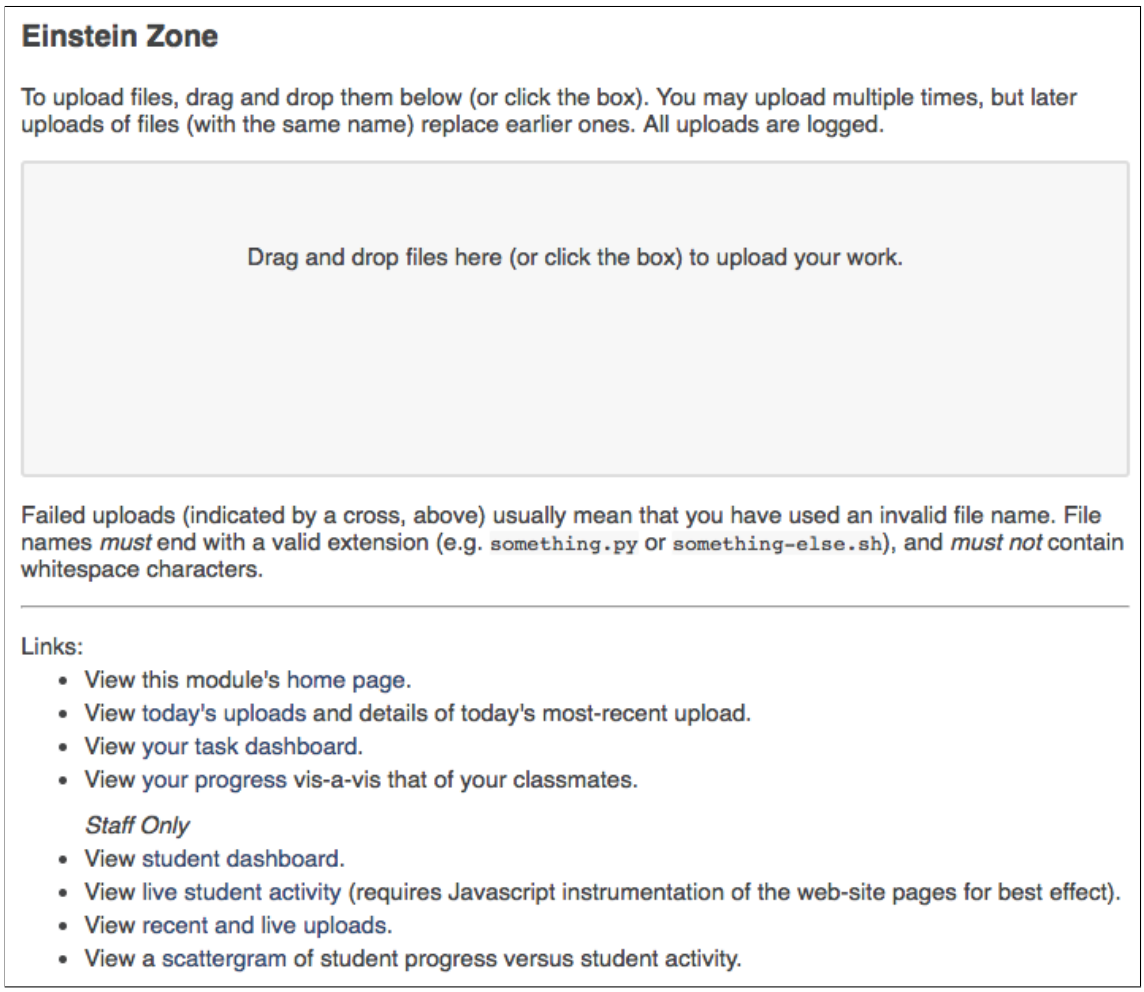


Figure 3.1: Screengrab from the Virtual Learning Environment for the Teaching of Computer Programming at Dublin City University

Table 3.2: Courses in DCU's School of Computing used in this thesis

Course	Title	Degree	Year	Semester	Language
CA116	Programming I	CA	1st	Fall	Python
CA117	Programming II	CA	1st	Spring	Python
CA114	Enterprise Computer Systems	EC	1st	Spring	Shell
CA177	Programming Fundamentals I	EC	1st	Spring	Python
CA277	Programming Fundamentals II	EC	2nd	Fall	Python
CA278	Programming Fundamentals III	EC	2nd	Spring	Python

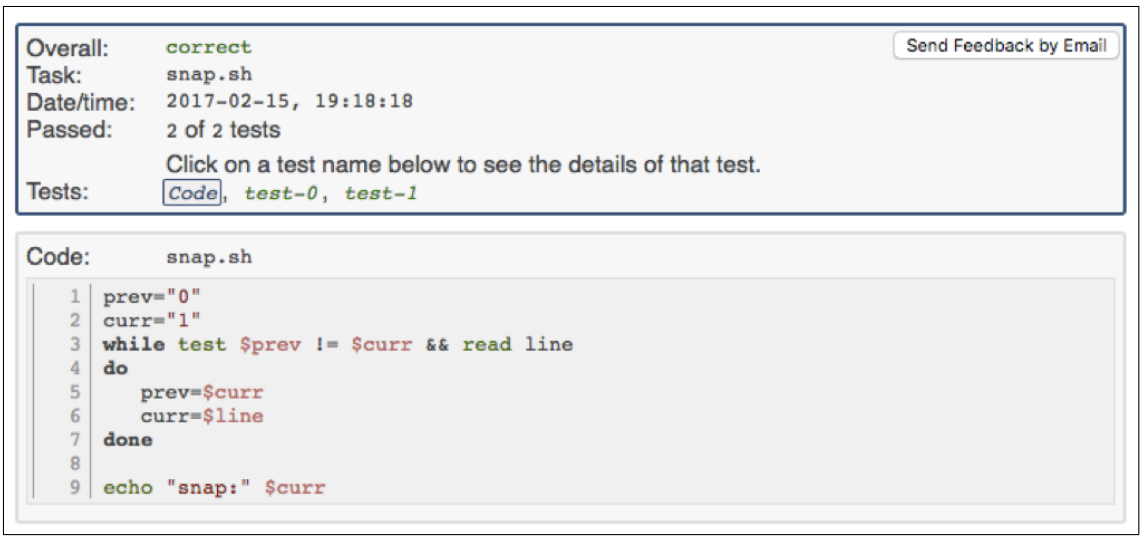


Figure 3.2: Instant Feedback Provided to the Student After Submitting a Program to the Automated Grading Assistant

3.1.2 Data from across the Whole University

In addition to log and interaction data from the custom VLE for learning computer programming, a range of other sources have been used to extract data on all first-year undergraduate students at the university. This was gathered in order to model their behaviour and chances of success at the university, and it includes the following:

- **Student Characteristics**: gender, date of birth, citizenship and domicile.
- **Prior Academic history**: prior-to-university test scores and prior academic history at the university.
- **Registration**: date of first registration and date of cancellation if applicable.
- **Clubs and Societies**: membership to any organisation run at the university.
- **Funding**: receipt of any SUSI (Student Universal Support Ireland) funding.
- **Library**: dates of borrowing instances (books) and dates and times of each occasion entering library building.
- **Computer resources**: occasions students use the on-campus computer labs.
- **Printers**: usage of printing services around campus.

- **Student Support (CRM)**: records of student interaction with the Student Support office for the school attended.
- **General VLE usage**: students at Dublin City University use Moodle as their main VLE and records associated with assignment submission were recorded and extracted.
- **WiFi**: Eduroam access logs of students logging in to different hotspots on campus.
- **Sports Centre**: records of students using the sports facilities on campus.

Approval for access to this confidential non-anonymised student data was granted by DCU Research Ethics Committee, reference DCUREC/2014/195.

3.2 Arizona State University

Arizona State University (ASU) is a public research university ranked number 1 in the U.S. for innovation, and dedicated to accessibility and excellence. ASU has five campuses spread across the Phoenix metropolitan area, and four regional learning centres throughout Arizona. ASU is one of the largest public universities by enrolment in the U.S. with 80,000 students attending classes across its metro campuses and 30,000 students attending online.

3.2.1 School of Computing, Informatics, and Decision Systems Engineering

Arizona State University's Web Programming Grading Assistant (WPGA) [2] was developed by the University to serve as a platform that connects the physical and the digital learning spaces in learning computer programming. This system enables the digitization, grading, and distribution of paper-based assessments. Further details regarding the rationale and the design of the platform can be found in [50]. All events

[2] https://cidsewpga.fulton.asu.edu/

(which mostly are students' clickstreams) are logged along with their timestamp. Examples of the clickstream data which is logged include logging in and out, clicking on a question to review, bookmarking a question, navigating through an exam, and taking of notes.

The data used in this thesis was collected from a classroom study conducted in a Data Structures and Algorithms course offered during the Fall 2016 semester. This class had a total of 3 exams and 13 quizzes. Among the 13 quizzes, only 6 were graded while the remaining 7 were recorded only for attendance (full credit was given regardless of the answers). There were 283 students enrolled in the class but only 246 (86.93%) were included in the study as those who dropped the course in the middle of the semester, did not take the three exams, or did not use the reviewing platform at all had to be removed. In the study presented in this thesis, we analysed *review actions* performed by students. A review action is an event where a student examines his or her graded answer. It includes reading the question, the answer, the assigned score and the feedback provided by the grader (see Figure 3.3 for an example). These *review actions* are collected via the web logs. We consider a student reads a question or answer when he or she clicked on the material resource.

3.2.2 Global Freshman Academy and ALEKS via EdX

In 2016, Arizona State University (ASU) launched the Global Freshman Academy (GFA) where they provide first-year university courses through the EdX platform allowing students to earn transferable ASU credits from anywhere. GFA makes university education available to anybody, from high school students to retirees going back to study at college. ASU currently offer 13 courses and our analysis will focus on two Mathematics modules: "College Algebra and Problem Solving I", and "Precalculus".

These courses leverage the Assessment and Learning in Knowledge Spaces (ALEKS) technology, which is a web-based artificially intelligent assessment and learning system owned by McGraw-Hill Education. This technology was developed at New York

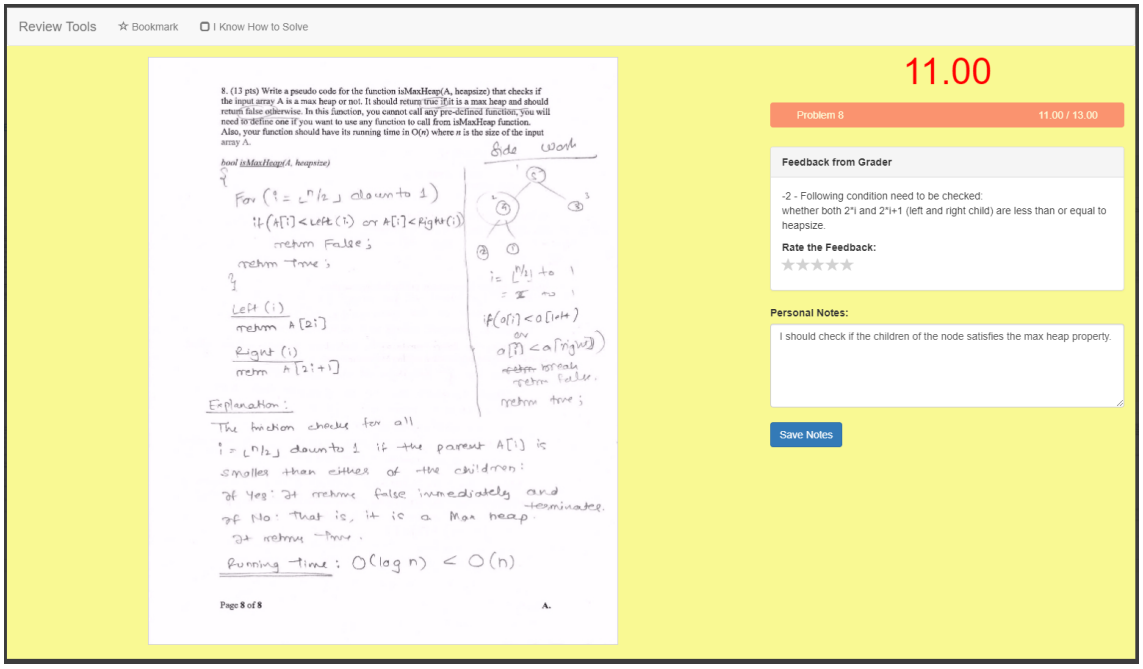


Figure 3.3: Web Programming Grading Assistant Platform When a Student Reviews Her Graded Response to an Exercise

Table 3.3: Some Math Courses at ASU's GFA in EdX

Course	Title
MAT117	College Algebra and Problem Solving I
MAT170	Precalculus

University and the University of California, Irvine supported by a National Science Foundation research grant. The two aforementioned Mathematics courses taught by ASU use this ALEKS technology.

ASU's GFA Mathematics courses combined with ALEKS technology is ASU's effort to get students prepared for college-level mathematics. The effectiveness and adaptiveness of this Artificial Intelligence (AI) tutoring systems have the potential to motivate and help students acquire these skills so their experience when they do reach college, is improved because of this grounding in mathematics.

An anonymized dataset of 15,000+ students learning on the two Mathematics courses in EdX with the ALEKS technology was collected between April 2016 and October 2017. Students were assessed continuously while navigating through

ALEKS and daily aggregates of the topics learned and retained were generated. We tracked 40,000+ assessments and 8+ million daily aggregates in this work. In addition, 5+ million transactions of students navigating through the concepts have been extracted from the EdX logs. Each timestamped transaction contains information on the student, the concept being studied and a learning state. The learning states are the following, and final states are determined automatically by the system:

- **L**: Initial state for each concept where a student reads the **L**esson
- **C**: Intermediate state where a student gets an exercise **C**orrect
- **W**: Intermediate state where a student gets an exercise **W**rong or Incorrect
- **E**: Intermediate state where a student asks for a working **E**xample
- **S**: Final state where a student has **M**astered a particular concept
- **F**: Final state where a student has **F**ailed to master a particular concept

In the next chapter we will describe how we used established machine learning techniques to model students' learning based on their digital footprints.

3.3 Feature Importances

In order to determine which features have more importance we used two approaches:

- The first one is an extra trees classifier, a type of forest that fits a number of randomized decision trees [41]. We utilized this technique for the predictive models developed for Dublin City University's computing courses, explained in more detailed in Chapter 4. We fitted an extra trees classifier with 250 estimators to measure the importance of each feature. Interestingly, we can observe how static features such as previous university scores or the entry-to-university Mathematics score are important in the first week of the semester while the dynamic programming work features and the effort students put in

increasingly gain importance throughout the semester relegating static features to the end of the ranked list of features.

- The second one is an ablation study, which refers to removing some feature of a model from the algorithm and seeing how that affects performance [65]. We leveraged this approach for determining the most meaningful features for all first-year students at Dublin City University, also explained in more detailed in Chapter 4. We excluded each of the feature from a linear model in order to measure how much variance the feature contain. The top features were considered the most meaningful ones.

Finally, in our projects we can reduce the number of features in our models by selecting the most meaningful parameters. In addition, we can also reduce this dimensionality by using techniques such as Principal Component Analysis (PCA) [56] which projects the data into a subspace maximizing the variance retained. These are areas of great interest in ML, either selecting important features or dimensionality reduction, but are out of the scope of this thesis.

Chapter 4

Modelling Students' Online Behaviour

4.1 Introduction

This chapter introduces a new research methodology to automatically detect students "at-risk" of failing a computer-based examination in computer programming modules (courses). By leveraging historical student data from previous cohorts, we built predictive models using students' offline (static) resources including student characteristics and demographics, combined with online (dynamic) resources using programming and behavioural logs. Predictions are generated weekly during the semester.

4.2 Context and Dublin City University Courses

Dublin City University's academic year is divided in two semesters with one week of inter-semester break in between. Semesters are comprised of a 12-week teaching or classes period, a 2-week study period and a 2-week exam period. Laboratory sessions and computer-based examinations are carried out during the teaching period. We have developed predictive models for a range of computer programming modules including the following:

- **Computer Programming I, CA116**: This course is a core and fundamental subject taught during the first semester of the first year of the honours Bachelors degree in Computer Applications. Students learn the fundamentals of computer programming, and how to write, run and debug their own programs. Students also learn the fundamentals of computational problem solving. Lectures are taught for four hours on three different days each week for 12 weeks. The course also involves four hours of supervised laboratory work on two different days each week using the Python language. A previous version of CA116 was taught using Java before it was redesigned for the 2015/2016 academic year.

- **Computer Programming II, CA117**: This course introduces first-year students to more advanced programming concepts, particularly object-oriented programming, programming libraries, data structures and file handling and is taught during the second semester of the first year. Students are expected to engage extensively in hands-on programming with the Python programming language. CA117's lectures are taught for four hours on three different days each week for 12 weeks and the course also involves four hours of supervised laboratory work on two different days each week using the Python language. Similar to CA116, a previous version of CA117 was taught using Java before it was redesigned. The current version with Python has been taught since 2015/2016 academic year. The course is a continuation of CA116, Computer Programming I, the introductory programming course.

- **Managing Enterprise Computer Systems, CA114**: This course equips first-year Enterprise Computing students with the basic skills necessary to administer modern enterprise operating systems and shows students how to manage Unix and Unix-like systems. Specifically, they study the Unix shell and work shell scripting programming exercises using tools like test, find or grep and concepts like loops, pipes and file handling. CA114's lectures are taught for two hours on two different days each week and the course also

involves two hours of supervised laboratory work each week using the Bash Unix shell and command language. This course has been taught for the past seven academic years since 2010/2011. Students work with the Bash Unix shell and the command language.

- **Programming Fundamentals II, CA277**: In this course second-year Enterprise Computing students learn to design simple algorithms using structured data types like arrays (known as lists in some programming languages) and dictionaries (also known as hash-maps) using the Python language. Students will be able to design simple algorithms using these structures, and write and debug computer programs requiring these data structures. Students will also learn how to write functions. This is a new course that has been taught for the first time the academic year 2016/2017.

- **Programming Fundamentals III, CA278**: This course teaches second-year Enterprise Computing students fundamental data structures and algorithms in computational problem solving. The material includes linked lists, stacks, queues or binary-search trees; and other techniques, like recursion. The language chosen is Python. CA278 is a continuing course of CA277. Lectures and labs are taught for four hours on two different days each week and is typically split between two hours of lectures and two of laboratory work. It emerged along with CA177 (Programming Fundamentals I) and CA277 for a need for Enterprise Computing students to have deeper computer programming skills in the workplace.

CA116 and CA277 are taught in the first semester (Fall). CA117, CA114 and CA278 are taught in the second semester (Spring). In all courses, students are assessed by taking two laboratory computer-based programming exams, a mid-semester and an end-of-semester assessment, during the teaching period. In CA278, instead of an end-of-semester lab exam, students demonstrate a working project. Each laboratory exam or demo contributes equally to their continuous assessment mark; 15% in

CA117, 25% in CA114 and 20% in CA278. Students are not required to submit their laboratory work for CA114 or CA277. In contrast, laboratory work count towards their final grade of the course for CA117 and CA278, both count as 10% of the overall grade for the course.

The university also provides in-lab peer-mentoring for some courses as well as the lecturer attending the laboratory sessions. In CA116 and CA117 around eight CA second-year students give tutoring support during laboratory sessions. In CA114 and CA278 a postgraduate student has been providing support to students during laboratory sessions. The automated grading platform, introduced earlier in Section 3, is currently used in a variety of programming courses across computing at DCU including CA116, CA117, CA114, CA277 and CA278. Students can browse course material, submit and verify their laboratory work.

4.3 Exploratory Data Analysis (EDA)

We know that students' digital footprints commence prior to their arrival at the university as demographics and GPA (CAO points) are collected at the time of application.

We analysed 950 first-year Computer Science (CS) entrants across a seven year period through the Leaving Certificate entry route. Early analysis showed a high correlation between the entry level GPA equivalent and first year final exams aggregate as shown in Figure 4.1. CAO (Cantral Applications Office) points can max at 600 and while there is no theoretical minimium, there is a minimum number of CAO points required for entry into each University course in Ireland and generally this is above 300 points. The Precision Mark shown on the y-axis in Figure 4.1 is the overall percentage aggregated across all subjects in first year, including computer programming modules.

In addition to Precision Marks, we looked at the number of enrolled students and pass rates for all modules in CS over the years. This was done to understand where students were having the most trouble in terms of module performance and, hence,

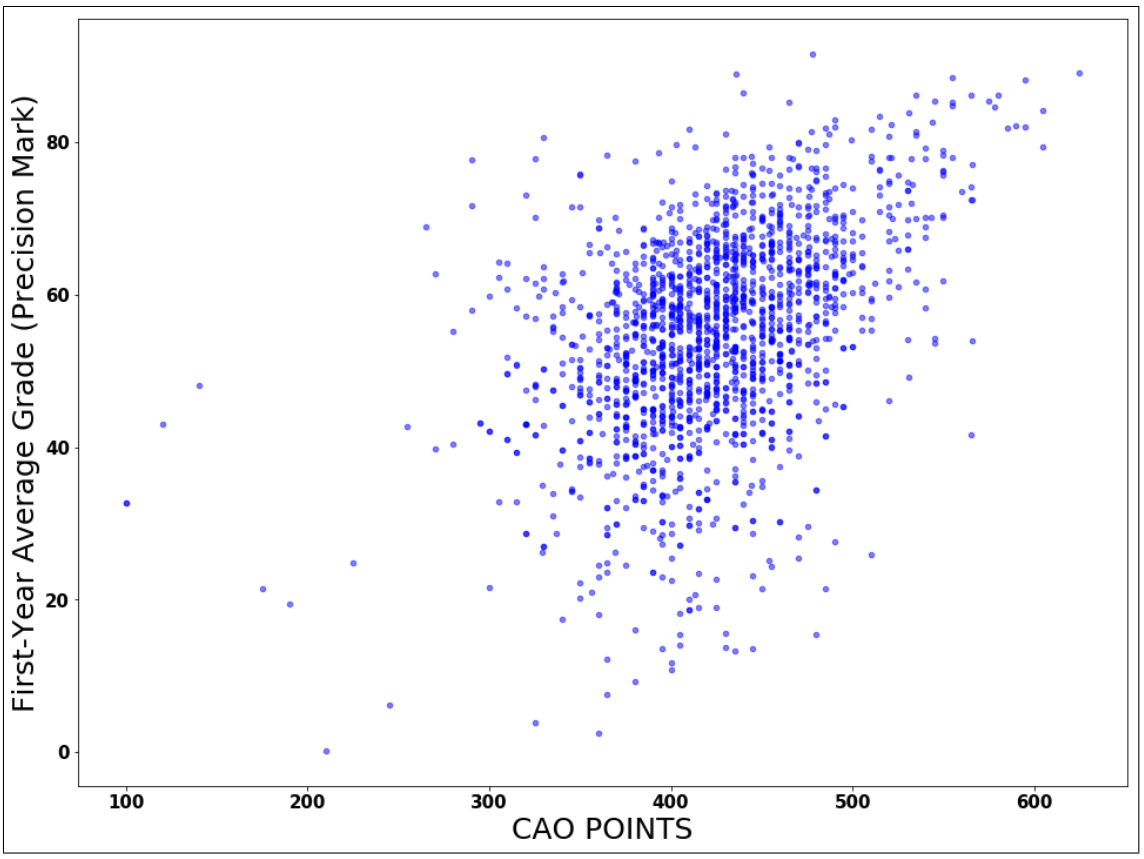


Figure 4.1: University Entry Points Correlated with First-Year Precision Mark for Computing Students at Dublin City University from 2013 - 2014 Academic Year to 2016 - 2017 Academic Year

where our work would have the most impact. For instance, Figure 4.2 shows the number of students enrolled in one of these courses, CA114, since it was introduced in the CS curriculum in 2010/2011 academic year.

Figure 4.3 shows the number of examinations taken in May with respect to the resit examinations which are taken in August, from 2009 - 2010 Academic Year to 2015 - 2016 Academic Year.

Figure 4.4 shows how CA114 is one of the courses with the highest failure rates in some past years and where students were having the most issues. This module along with the others mentioned earlier in first and second year were then added to our studies.

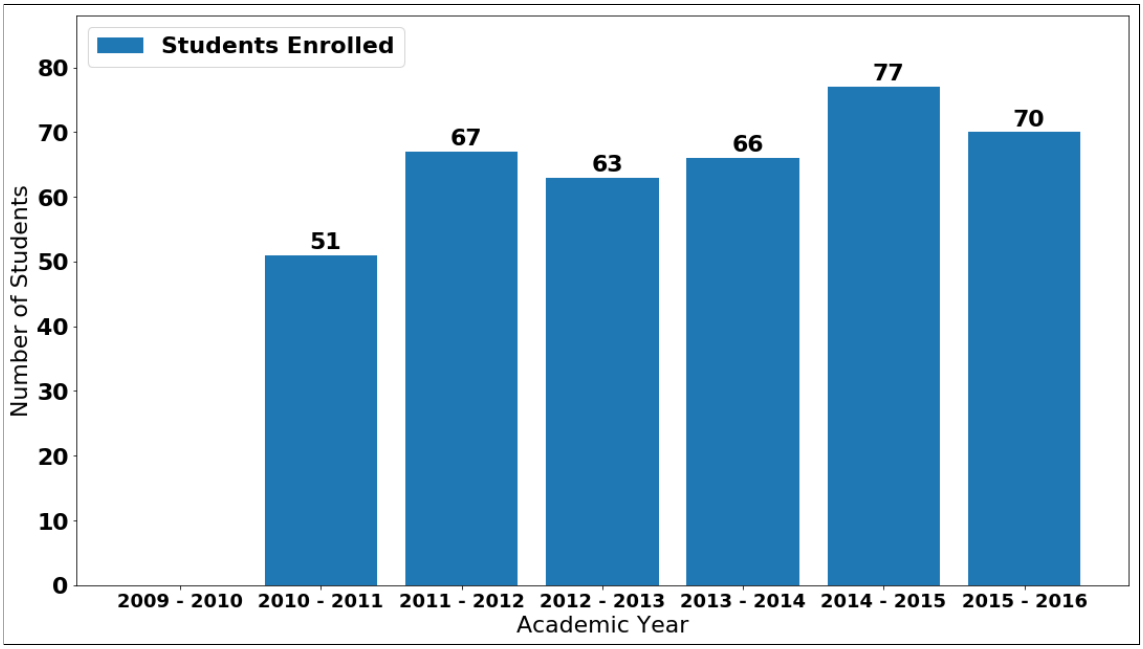


Figure 4.2: Number of Students Enrolled in CA114 over from 2009 - 2010 Academic Year to 2015 - 2016 Academic Year

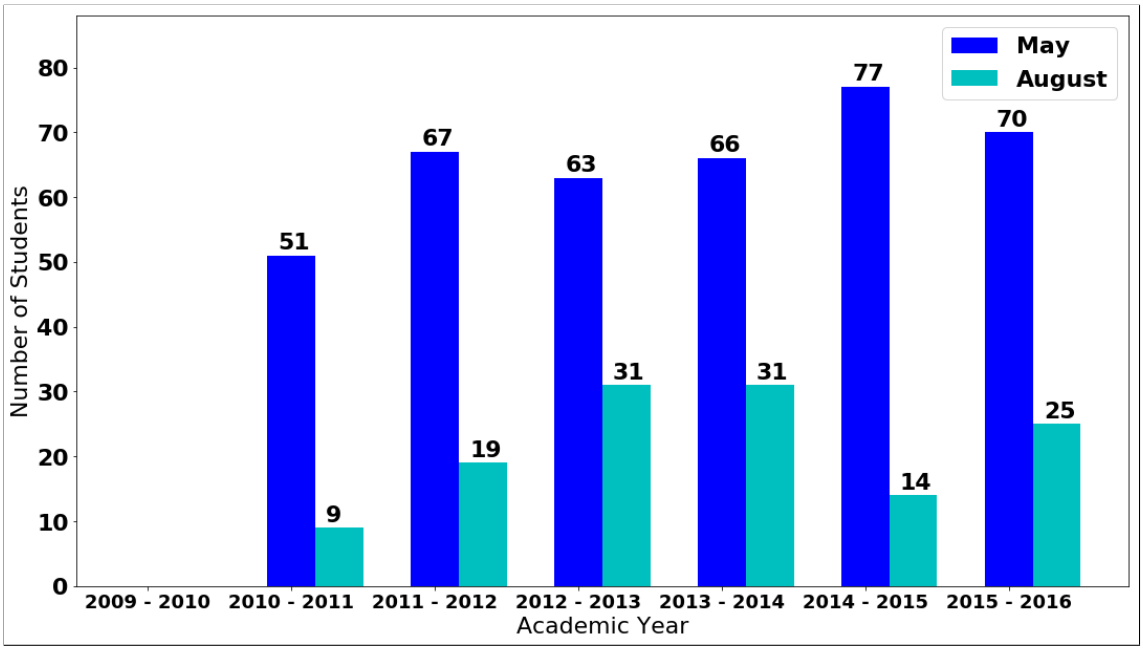


Figure 4.3: CA114's Numbers per Examination over from 2009 - 2010 Academic Year to 2015 - 2016 Academic Year

4.4 Data Processing and Feature Engineering

For each course, a number of features were extracted from the data in a weekly basis. A combination of static and dynamic student features was used for a weekly

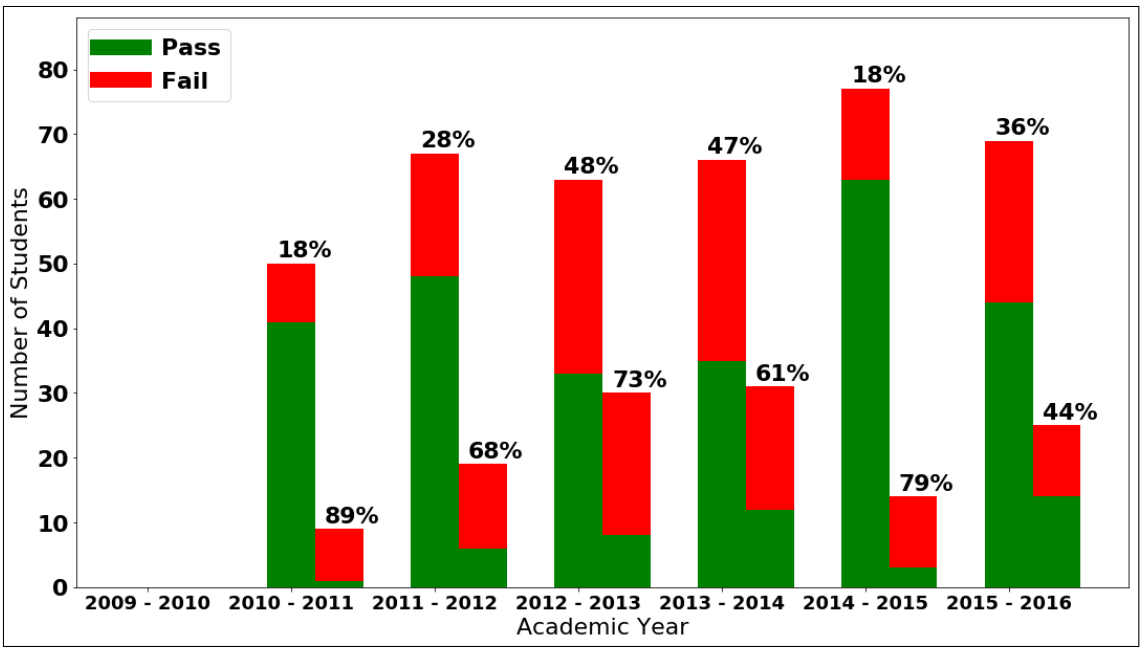


Figure 4.4: CA114's Failure Rates Per Examination from 2009 - 2010 Academic Year to 2015 - 2016 Academic Year

learning function we will introduce in the next section. A set of **static features** was extracted before the start of the semester for each course and each student based on their characteristics and prior academic performance. Then, each week, a set of **dynamic features** was collected for each student by building engagement and progression features based on their interactions and submissions to the platforms. The data sources we leverage in order to model student interaction, engagement and effort in computer programming courses are **student characteristics**, **prior academic history**, **interaction logs** and **programming submissions**, as explained in Chapter 3.

The following set of **static features** were extracted before the start of the semester for each course and student:

- **Student characteristics:**
  - Age in years based on their date of birth and registration date.
  - Travel distance in Km from home to university based on their domicile.
  - Route to university: Irish Leaving Certificate, Athletic scholarship, Dis-

advantaged background, Mature entry, etc.

- **prior academic performance:**
  - Irish CAO points and Leaving Certificate exam scores (equivalent to High School GPA and SAT exams in the US)
  - Prior academic history at the university.
  - Final laboratory exam grades from prior courses in CS.

Then, each week, a set of **dynamic features** were extracted for each student based on raw log data, interaction events for students accessing material and corresponding computer programming submissions. For instance, in [69] it was found that high level of VLE activity was a good indicator and, in particular, evening activity was a indicator of good performance. See the following dynamic features extracted:

- **programming effort:**
  - Computer programming laboratory work completed that week: percentage of correct exercises on that week's labsheets.
  - Cumulative computer laboratory work completed since the start of the semester.
- **engagement:**
  - Lab attendance: whether the student attended the laboratory sessions or not.
  - Time spent on the online platform.
  - Ratio of during-laboratory to non-laboratory time accesses.
  - Resources clicked with respect to all resources made available each week.
  - Average time of the day the course material is accessed.
  - Average lapse time between a resource being made available and the student accessing it for the first time.

- Ratio of on-campus to off-campus accesses based on IP address.
- Ratio of weekday to weekend accesses.

Table 4.1 lists the features with an associated short name that we will use for graphs and tables in the remainder of this chapter.

Table 4.1: List of Feature Names and associated Short Names

Feature name	Short name
Travel distance to university	Distance
Irish CAO points (High School GPA)	CAO points
Leaving Certificate Math Exam Score (SAT exams)	Math LC
Average grade on previous formal assignments	Avg. Grade
Laboratory assignment *Course Year*	Exam *Course Year*
Computer programming work completed in week x	Program Correct Wx
Cumulative programming work completed in week x	Cum. Programs Wx
Ratio of on-campus to off-campus accesses in week x	Campus Rate Wx
Ratio of weekday to weekend accesses in week x	Week Rate Wx
Resources clicked over all resources made available in week x	Coverage Wx
Laboratory attendance week x	Lab Attendance Wx
Time spent on the platform in week x	Time Wx
Ratio of during-lab to non-lab time accesses in week x	Lab Access Wx
Average time of the day material is accessed in week x	Hour Access Wx
Average lapse time to access resources in week x	Checking Wx

At this point in the thesis, we will focus on just one of the modules in order to make the research methodology understood better. The module we choose is CA116, Programming I, for first-year Computer Applications students.

4.5 Feature Exploration and Correlations

In order to identify the predictive power of the various features in our student model, we measured the linear and non-linear relationships between their values

and our predicted target, the next laboratory exam results. We used the Pearson and Spearman correlation coefficients and p-values (to indicate the probability of uncorrelation and the null hypothesis to be incorrect).

For instance, in the last week of the teaching period (week 12), see Figure 4.5, the cumulative programming work and the grade are highly correlated (correlation coefficient of 62% with an associated p-value less than 0.01).

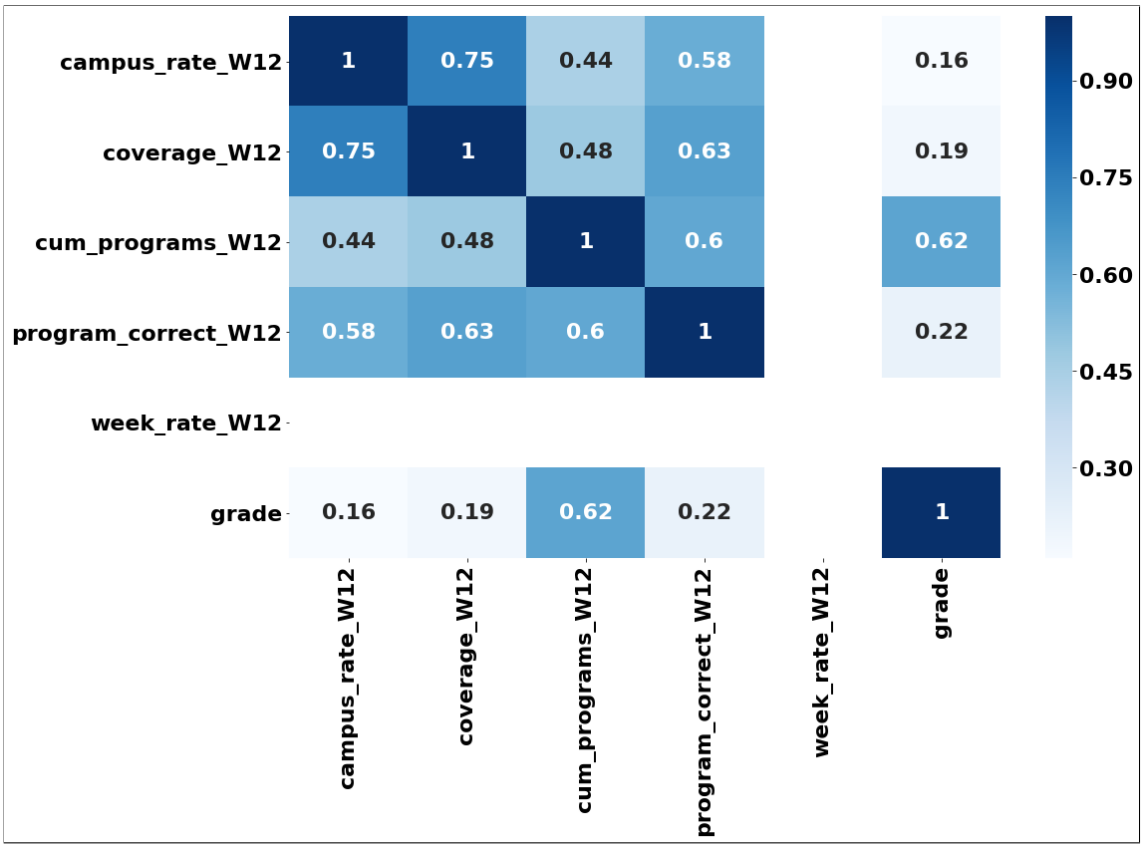


Figure 4.5: Correlations for CA116 2016/2017's Features in Week 12

For each week in the academic year 2016/2017, we measured the correlation among features of the same week for CA116, see Figure 4.6, the labels for each graph are in the same order as in Figure 4.5: Campus Rate, Coverage, Cum. Programs, Programs Correct, Week Rate (all of them for each week) and the Grade (the target of our predictions and the parameter we use to calculate the correlation coefficients).

This analysis confirms the predictive power of our features and the programming weekly and cumulative progress features increasingly gain importance throughout the semester as students put more effort into the module and the module learns

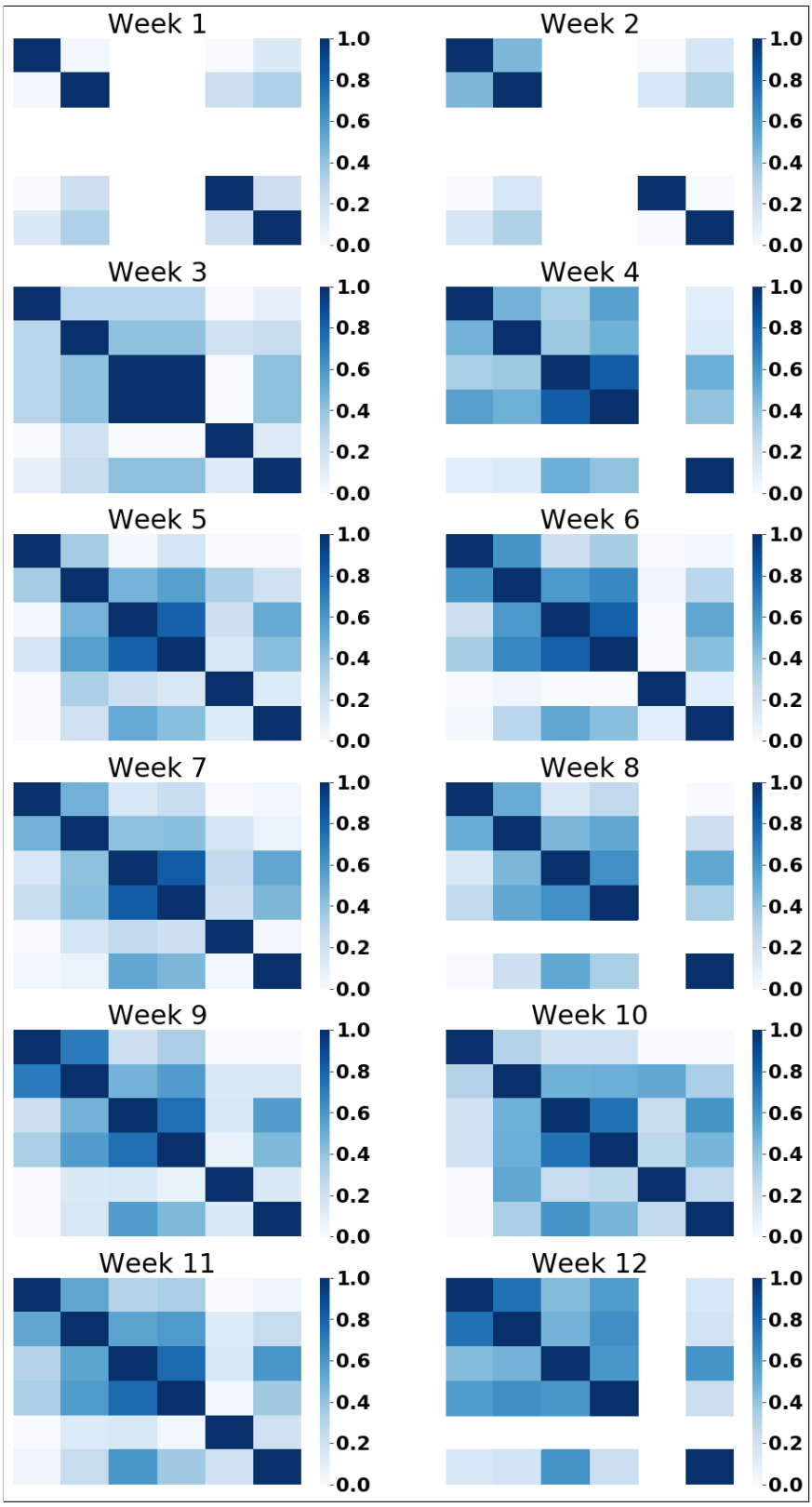


Figure 4.6: Correlations for CA116 2016/2017's Features Every Week

more about each student because it has more data on each student to learn from.

4.6 Splitting Data between Training, Validation and Testing

Over the last few years, at Dublin City University, we have been developing predictive models to identify students struggling or having issues with course material that they are studying. We do this by training these models with past student data from previous cohorts of students and running these in pseudo real-time (running them every week) with incoming new student data on the present cohort of students based on their engagement activities during the previous week.

For the course CA116 and the academic year 2018/2019, we trained weekly predictive models leveraging student data from student cohorts in 2016/2017 and 2017/2018. See Figure 4.2 for details. This module had three laboratory examinations during the semester which took place during weeks 4, 8 and 12. The results of those exams will be the target of our predictions. The proportion of students that passed vs. failed in the training and validation sets is roughly the same so we have a balanced training set.

Table 4.2: CA116 Split between Training, Validation & Test sets

Academic Year	Number of Students	Data Use
2016/2017	126	Train & Validation
2017/2018	156	Train & Validation
2018/2019	130	Test

We use a threshold of 40% to consider when a student passed an examination. This is the minimum grade in our university to pass a module. The target was to predict whether each student would pass or fail their next laboratory exam, i.e. to get more or less than 40%. Programming modules are quite dynamic and programming laboratory exercises vary considerably from year to year. However, the underlying programming concepts and knowledge being taught should remain the same.

4.7 Model Selection

4.7.1 Empirical Risk

A set of binary classifiers, one per week, were built to predict a student's likelihood of passing or failing the next computer-based laboratory exam based on their data. CA116 had three laboratory exams every semester so in that case, to clarify, classifiers from week 1 to 4, were trained to predict the laboratory exam outcome (pass or fail for each student) on week 4; from 5 to 8, the laboratory exam outcome on week 8; and from week 9 to 12, the end-of-semester's laboratory exam outcome in week 12.

At a given week, the dynamic features mentioned above were extracted from that week's activity log and programming submissions. Then, every week, a classifier was built by concatenating the static student data, the dynamic features from previous weeks' classifiers and that week's dynamic ones in order to account for each student's progression throughout the course.

In terms of implementation, the empirical error minimization approach was employed to determine the learning algorithm with the fewest empirical errors from a bag of classifiers C [33]. The bag of classifiers consists of the following learning algorithms:

(a) Logistic Regression: statistical algorithms that models the probability of a class and transforms that into a class using the logistic function.

(b) Decision Tree: interpretable model where a tree is built by splitting the training data using the classification features.

(c) Random Forest: ensemble classifier that fits a number of decision tree classifiers and uses averaging to improve accuracy and control overfitting.

(d) K-Neighbors (k-NN): algorithm that memorizes the labelled data during training and makes a decision on classification by looking at the k closest training examples. k is a hyper-parameter.

(e) Multilayer perceptron (MLP): feedforward Artificial Neural Network (ANN) with at least one hidden layer and each node uses a nonlinear activation function.

These models were trained every week with the training data and evaluated on the validation data for several scoring metrics including receiver operating characteristic area under the curve (ROC AUC), accuracy (see Figure 4.7), F1 score (see Figure 4.8), precision and recall. In addition, instead of just taking the learning algorithm with the lowest empirical risk or the highest metric (namely accuracy or F1-score), we also looked at these metrics per class. Generally, the results on the next laboratory exam, our target variable, is quite imbalanced as on some courses there are more students that pass than fail the exams. The resulting accuracy of a learning algorithm could be misinterpreted if we weight the predictions based on the numbers per class. Our goal is to identify weak students as we would rather classify students "on the edge" as likely to fail than not flagging them at all and miss the opportunity to intervene and help them.

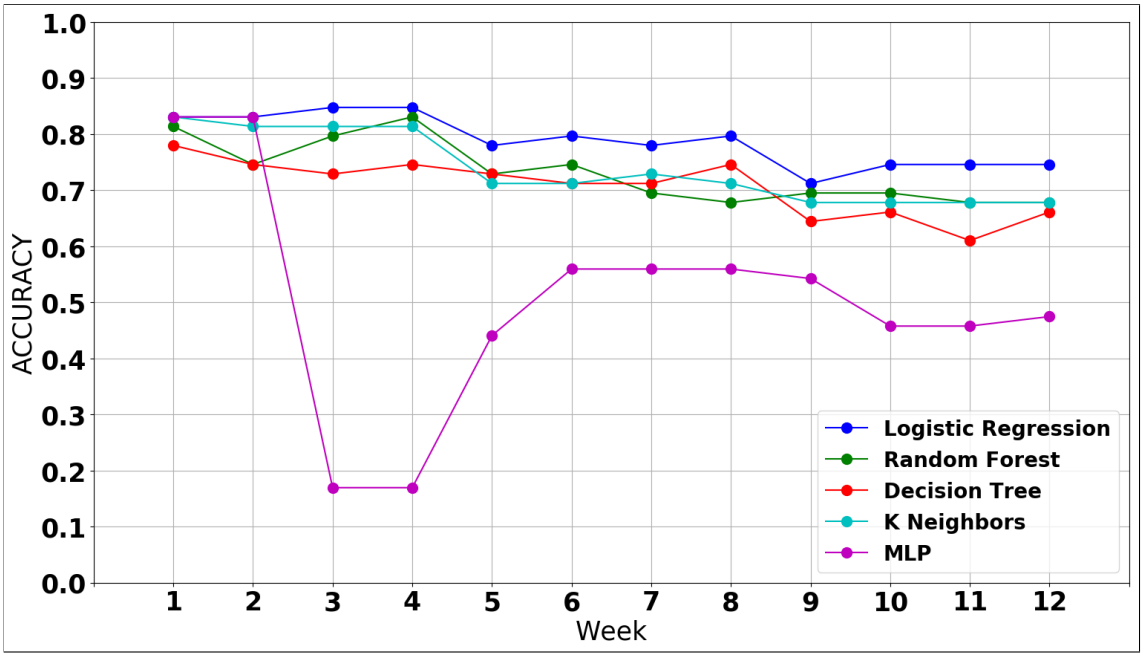


Figure 4.7: Empirical Risk for CA116 for the Training Data using Accuracy

We should note that we only have two years of archival data from previous student

cohorts and thus not many samples of past student data for a machine learning algorithm. Hence, most classifiers perform very similarly.

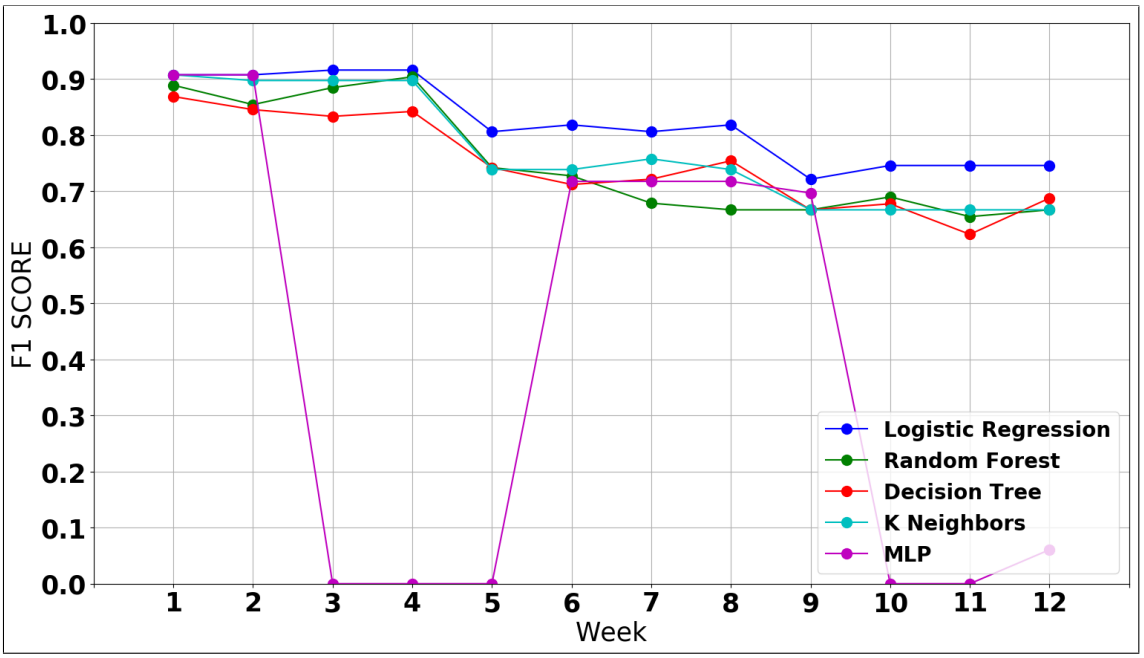


Figure 4.8: Empirical Risk for CA116 for the Training Data using F1-Score

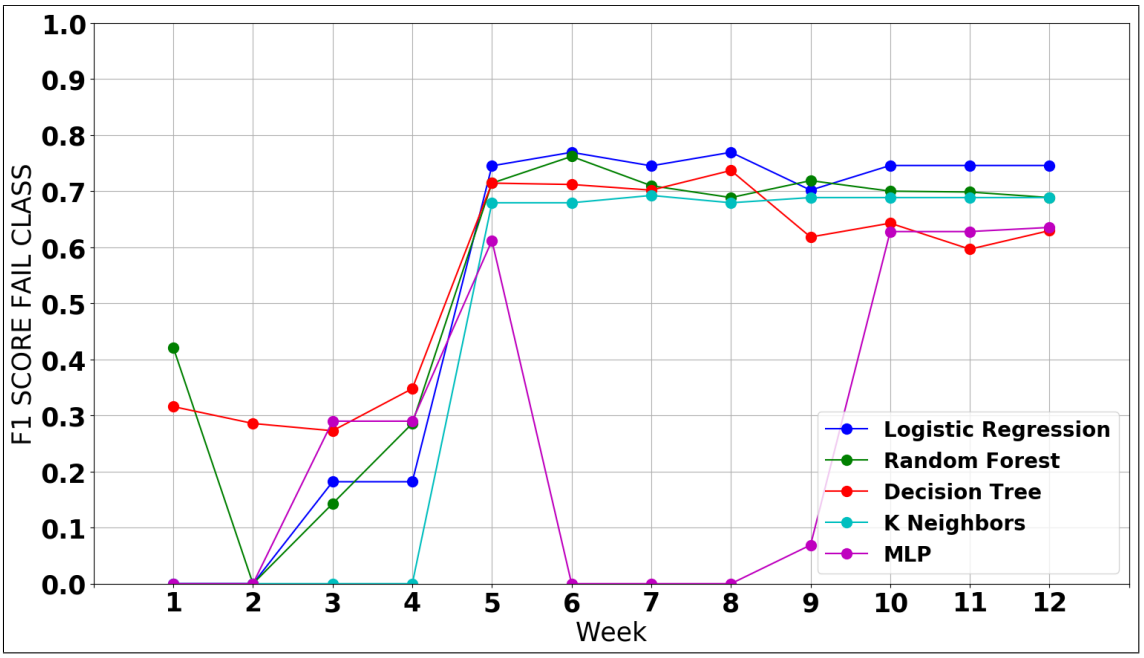


Figure 4.9: Empirical Risk for CA116 for the Training Data using F1-Score for the Fail Class Only

Following this approach, we chose to use the learning algorithm which minimized

the empirical risk on average for the 12 weeks which was then used in deployment for each weekly classifier. Figure 4.8 and Figure 4.9 shows how some of these classifiers are not doing a good job at detecting students that are likely to fail. Classes are typically imbalanced as there might be more students passing a particular exam than failing it. Hence, F1-Score is an appropriate metric that shows the MLP classifier is not learning at all and predicting all students would pass for some of the weeks. Random Forest is the chosen algorithm from the bag of classifiers as it minimized the empirical risk on average for the 12 weeks and keeps a good balance for both classes. Random Forest has proven to outperform other classifiers by mining student data for predicting performance [68]. Other algorithms in our bag of classifiers such as MLP might have poor initializations and does not reach a global minima.

4.7.2 Hyperparameter Optimization

In the hyperparameter optimization or tuning phase we choose the optimal hyperparameters for a learning algorithm. A hyperparameter is a parameter whose value is used to control the learning process. The search for these parameteres is guided by some performance metric, typically measured by cross-validation on the training set or evaluation on a held-out validation set [27]. The two most common ways of performing hyperparameter optimization are:

- Grid Search: an exhaustive searching through a manually specified subset of the hyperparameter space of a learning algorithm
- Random Search: replaces the exhaustive enumeration of all combinations by selecting them randomly

In our study, we chose a Random Forest classifier and, for instance, one of the hyperparameters that has to be tuned is the number of trees. So, for every week, using the validation data we held out, we optimized these hyperparameters for a Random Forest classifier using Grid Search. Then, we stored these weekly learned

models to be used with incoming student data. Figure 4.10 shows how this trained model performs on the validation data we held out.

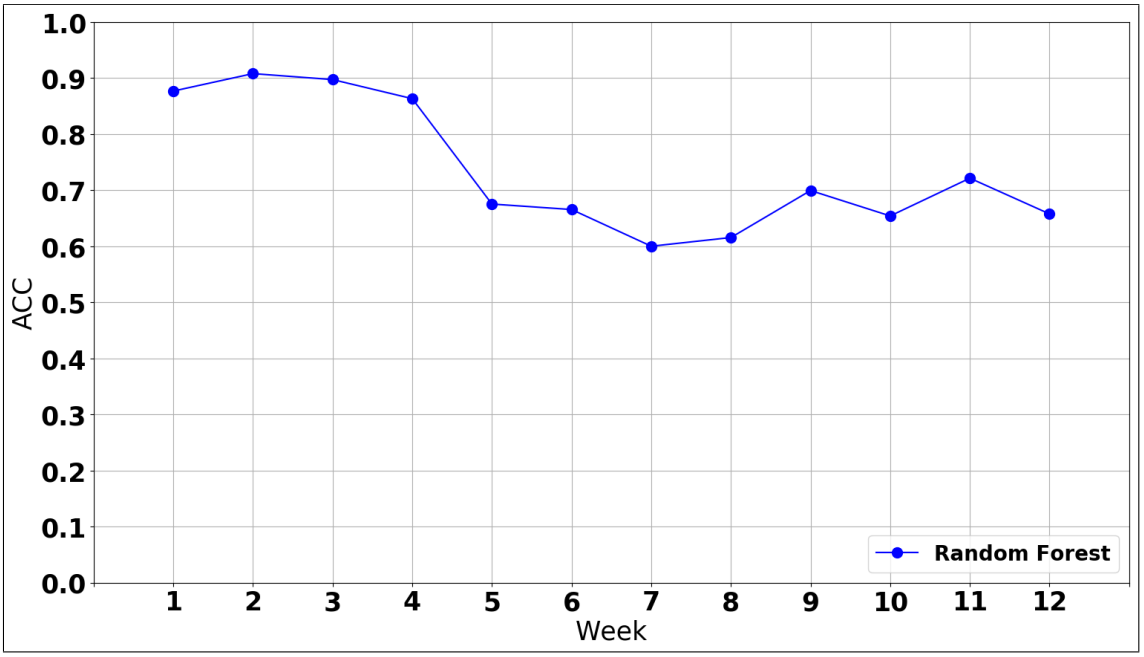


Figure 4.10: CA116's Validation Data's Accuracy after Hyperparameter Tuning

4.8 Predicting which incoming students are "at-risk"

For the incoming cohort of 2018/2019 students taking the CA116 module, we leveraged the models trained on previous cohorts to predict whether each incoming student would pass or fail the next laboratory examination, every week, and the associated probability of doing so. A summary report was sent to the lecturers of the module and other faculty associated with the classes. See Figure 4.11 for an anonymized example of the feedback on (in this case just 3) student predicted outcomes. This information was also posted on our web application accessible to lecturers. Reports were also posted on a web application where the retrospective analysis, the feature values tabulated for each student and more analysis in detail could be found, similar to [34].

Student	Fail / Pass Prediction	Fail Prediction Confidence
John Doe	Pass	26.95%
Jane Roe	Fail	69.27%
Johnny Smith	Pass	27.03%

Figure 4.11: Snapshot of Anonymised Predictions for a Sample of 3 Students

4.9 Re-visiting RQ1: Accuracy of Predictive Modelling

In section 1.1 the first research question in the thesis was introduced and it is re-stated here for convenience:

RQ1: When working with new cohorts of University students about whom we have little historical interaction data, how accurate are the traditional Predictive Analytics models when used with generic static and dynamic student data features, in identifying those students in need of assistance in computer programming courses?

We analysed the preliminary results obtained and their impact on the first research question proposed in this chapter. In order to evaluate how the predictions performed in terms of accuracy, we compared the corresponding weeks' predictions with the actual results of the three laboratory exams that took place in weeks 4, 8 and 12 for CA116. This meant we were able to investigate how our predictions worked with respect to the actual 2018/2019 students' grades and the details are shown in Table 4.3. As the semester progresses, the pass rates for the three laboratory exams reduce. We only show the weeks where there was a laboratory exam in this case, but we could also have a look at the rest of the weeks and see how our model performed on new incoming data.

For each of those exam weeks we created a confusion matrix with the expected pass/fail and the actual results by looking at the true positives, true negatives,

Table 4.3: CA116 Prediction Metrics including passing rates and at-risk rates

Exam Week	Pass Rate	Predicted At-risk	Accuracy	F1 Score	Precision	Recall	Correlation Coeficient
4	84%	1%	0.85	0.92	0.85	1.00	57%**
8	64%	58%	0.72	0.74	0.94	0.61	69%**
12	47%	60%	0.81	0.78	0.84	0.72	72%**

** $p - value < 0.001$

false positives and false negatives and from this we are able to calculate accuracy, precision, recall and F1-score as shown in the Table. Also, for each laboratory exam, we looked at the passing rate to compare it with the percentage of students at-risk predicted by our models. These metrics were also shared with the lecturer. The evaluation metrics are now further explained:

- Accuracy: (True Positives + True Negatives) / Total
- Precision: True Positives / Predicted Condition Positive = True Positives / (True Positives + False Positives)
- Recall: True Positives / Condition Positive = True Positives / (True Positives + False Negatives)
- F1 Score: 2 * (Precision * Recall) / (Precision + Recall)

Figure 4.12 shows how accurately the predictions worked for each week of the semester using the F1 measure (not only the exam weeks). We chose F1 as the main evaluation metric as classes might be imbalanced. Generally, the number of students that pass is not the same as the number of students that fail and we can not rely on accuracy.

In addition to the above, we looked at the probability associated with failing the laboratory exam that the classification algorithm gives us and linearly and non-linearly correlated that with the actual result the students obtained. This allows us

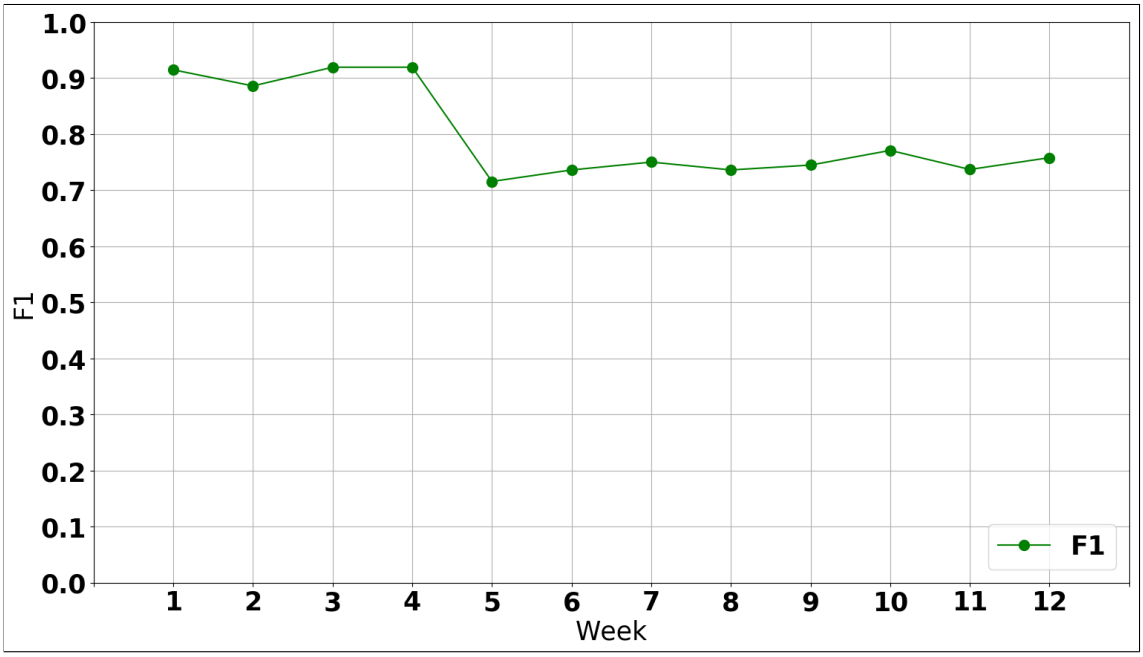


Figure 4.12: Evaluation using F1 for CA116's Incoming 2018/2019 Cohort Shown Weekly

to have a deeper look at the confusion matrices for each of the exam weeks.

Figure 4.13 shows the confusion matrix for Week 4 while Figure 4.14 shows the confusion matrix for Week 8 and Figure 4.15 shows the confusion matrix for Week 12.

		Predicted Condition		
	Total Population	**Predicted Condition Positive**	**Predicted Condition Negative**	
True Condition	Condition Positive	True positive 108	False Negative (Type II error) 0	**Recall 100.00%**
	Condition Negative	False Positive (Type I error) 19	True negative 1	
	Accuracy 85.16%	**Precision 85.04%**		**F1 91.91%**

Figure 4.13: Confusion Matrix for Week 4 for CA116's Incoming 2018/2019 Cohort

Condition Positive means students passed the examination and Condition Negative means students failed the examination. Predicted Condition Positive means students were predicted as they were going to pass the examination and Predicted Condition

		Predicted Condition		
	Total Population	**Predicted Condition Positive**	**Predicted Condition Negative**	
True Condition	Condition Positive	True positive 46	False Negative (Type II error) 31	**Recall 59.74%**
	Condition Negative	False Positive (Type I error) 2	True negative 41	
	Accuracy 72.50%	**Precision 95.83%**		**F1 73.60%**

Figure 4.14: Confusion Matrix for Week 8 for CA116's Incoming 2018/2019 Cohort

		Predicted Condition		
	Total Population	**Predicted Condition Positive**	**Predicted Condition Negative**	
True Condition	Condition Positive	True positive 36	False Negative (Type II error) 17	**Recall 67.92%**
	Condition Negative	False Positive (Type I error) 6	True negative 54	
	Accuracy 79.65%	**Precision 85.71%**		**F1 75.79%**

Figure 4.15: Confusion Matrix for Week 12 for CA116's Incoming 2018/2019 Cohort

Negative means students were predicted as they were going to fail the examination. Moreover, True Positives means students passed the examination and were predicted to do so. True Positives means students failed the examination and were predicted to do so. False Positives means students were predicted or expected to pass the examination but in reality they failed. False Negatives means students were expected to fail the examination but in reality they passed.

The analysis carried out on CA116, as a usecase for this chapter[1], showed we were successfully able to:

(i) gather student data about the student's learning progress by combining static with dynamic information regarding their characteristics, prior academic re-

[1]The code for this work has been made available as a GitHub repository at https://github.com/dazcona/edm-modelling

sults, behavioural analysis and programming effort

(ii) leverage that digital footprint for predicting how new incoming students are likely to perform reaching a usable accuracy

In short, predictions worked quite well and we could automatically distinguish in a better way who is going to pass or fail as the exams in weeks 4, 8 and 12 were getting closer. We showed that extracting limited data from students learning how to program, we can develop accurate predictive models to identify student in need of support using traditional Machine Learning techniques. Hence, **RQ1** has been answered. Emails with feedback were sent to students which might have changed their behaviour, this is further explained in Chapter 6. This approach has been deployed in a variety of programming modules described at the beginning of this chapter and their lecturers were notified weekly with this type of information.

4.10 Extra: Retrospective Analysis on Reviewing Behaviours at ASU

During my stay at Arizona State University, we collaborated with the School of Computing, Informatics, and Decision Systems Engineering which is part of the Ira A. Fulton Schools of Engineering. In this work, we developed predictive models based on students' reviewing behaviours in a Data Structures and Algorithms course during the Fall 2016 semester. The platform where we extract our data is ASU's WebPGA educational technology that students used to review their graded paper-based assessments. The system is further described in Chapter 3.

The instructor administered a total of 3 exams and 13 quizzes throughout the semester. Among the 13 quizzes, only 6 were graded while 7 were for attendance only (full credit regardless of the correctness of answers). A total of 283 students were enrolled in the course. However, in this study, only 246 (86.93%) students were included as we excluded those who dropped out of the course in the middle of the

semester, did not take the three exams, or did not use the research platform. In this study, review actions performed by students were analyzed. A review action is an event where a student examines his or her graded answer.

Predictive models were trained after extracting patterns and tested with the goal of identifying students' academic performance and those who might be in need of assistance. The results of the retrospective analysis show a reasonable accuracy. This suggests the possibility of developing interventions for students, such as providing feedback in the form of effective reviewing strategies.

4.10.1 Data Processing

The students were labelled according to their overall academic performance. To achieve this, we computed the average of the three exams of each student (see Fig. 4.16 for the distribution). Using Jenks natural breaks classification method [55], the break-point of 77.60% was obtained and used as a threshold to divide the students into two groups, namely *higher performers* and *low performers*. This was done as only a few students fail the course on average and mastering the subject is only considered if students get a high grade.

4.10.2 Features

A student's digital footprint is shaped to combine the different modalities of student data and leverage that information to analyse the behaviour of the students on these courses using Artificial Intelligence techniques. In this study, logs captured by WebPGA as students browse through their digitized paper-based assessments have been leveraged to model their reviewing behavior.

First, a set of features was extracted based on the students' interaction within the system and the different actions they performed. For each assessment, the following were gathered:

1. their grades

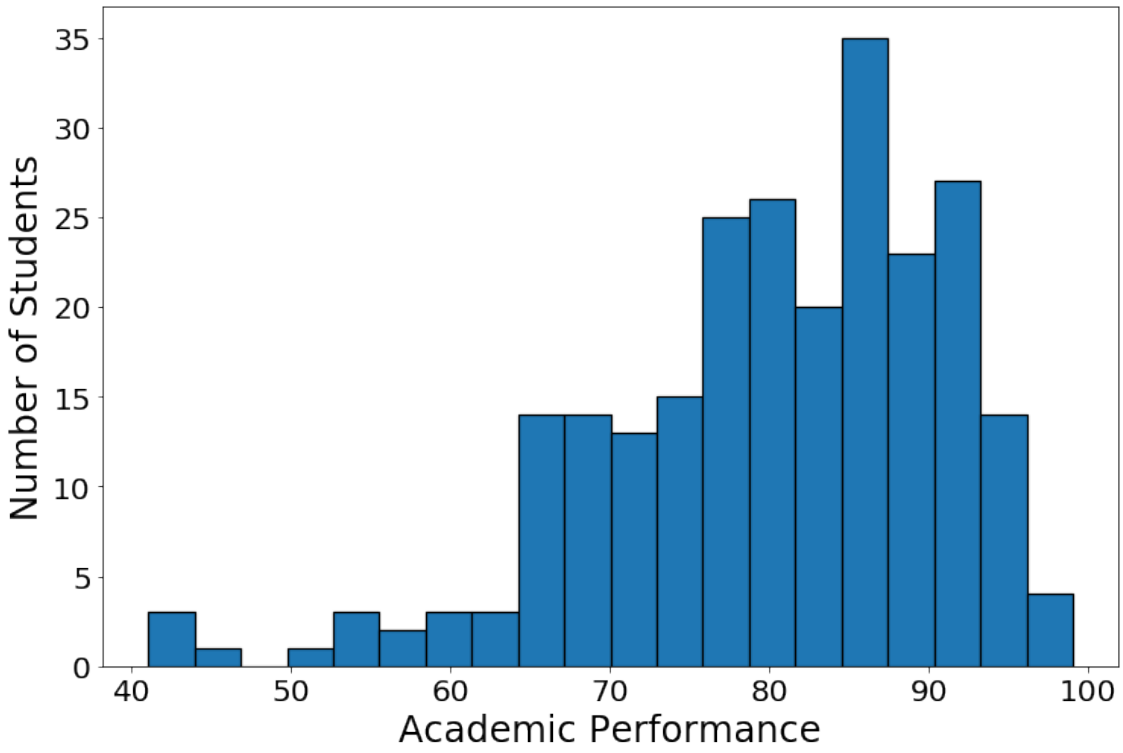


Figure 4.16: Distribution of Students' Academic Performance for Computing Students in a Data Structures and Algorithms Course at Arizona State University

2. whether they reviewed the assessment or not

3. when they reviewed it for the first time (after being posted online)

General engagement features were also collected. This includes:

1. number of distinct days when the system was used by each student

2. the number of interactions or how many assessments were reviewed per student

The predictive power of these features is measured by correlating their values with the cumulative exam average (target). Table 4.4 shows a few of those features and the corresponding correlation coefficient.

Students were more likely to obtain a better score on the average exam grade as they:

1. reviewed more assessments; or

2. accessed the system regularly on different days; or

Table 4.4: Feature correlations with the cumulative exam average

Feature name	Correlation coefficient
Number of assessments covered	34%*
Average time to review	−0.14%
Distinct Days	28%*
Distinct Actions	12%
Number of web interactions	24%*
Quiz 6 Mark	38%*
Quiz 10 Mark	35%*

* $p - value < 0.01$

3. obtained higher scores on their quizzes.

On the other hand, the later the student reviewed the assessments (on average), the lesser the chance he or she had of getting a high grade. This analysis indicates the high correlation of some of the features such as the number of assessments covered or the number of distinct days. In contrast, other parameters such as whether the students reviewed specific assessments or how long it took them to review particular assessments were individually not correlated with the average grade target.

4.10.3 Classification and Regression Modelling

A bag of learning algorithms was trained retrospectively using the students' digital footprint and their observed behavioural patterns. We analyzed their power to predict the students' performance and their generalizability. We used cross-validation to train and test on the same dataset. The target was to predict whether a student will score above or below the threshold of each cumulative exam average. Note that the cumulative exam averages (our target for each period) include: (1) the first exam score before it took place, (2) an average between the first and second exam before the second exam, and (3) an average of the three exams before the third exam was taken by students. The threshold, 77.60%, was used to divide high and

low performers, which is derived in Section 4.10.1.

The three exams divided the entire semester into four periods, namely *Before Exam 1, Exam 1 - Exam 2, Exam 2 - Exam 3, and After Exam 3*. For each period, a learning function is trained to predict a student's likelihood of scoring above or below the performance threshold on their cumulative average grades. For instance, the first classifier was trained to predict the first exam outcome (above or below the threshold), the second one was to predict the average outcome between the first two exams, and so on. In a given period, the features mentioned above (Table 4.4) were extracted from the student's interactions with the system along with their reviewing patterns. A classifier was built by concatenating all the features from previous assessments, such as scores and reviewing times.

Table 4.5 shows how more features were concatenated as students were being assessed throughout the semester. The percentage of students below the threshold was also checked. It shows that the two target classes (above and below) were balanced.

Table 4.5: Number Features per period and Students below the Threshold

Period	No. of features	Students below threshold
Before Exam 1	15	160 (65.04%)
Exam 1 - Exam 2	33	153 (62.20%)
Exam 2 - Exam 3	51	158 (64.23%)

In terms of the features' importance, their weights were plotted in Figure 4.17 using a heatmap. The general engagement features, such as the number assessments reviewed by students or the number of distinct days students logged in, were used individually in the model. Their weights were calculated per period and plotted on the graph. In addition, features developed which were specific for each assessments were grouped (*mark or score*, *whether they reviewed these assessments* and *how early they reviewed them*) into three single parameters: *Mark*, *Reviewed* and *Time*. Those three parameters aggregated the importance for each of those categories. For

instance, the features that capture the time students review each assessment for the first time were clustered into one parameter, Time, that adds up the weights of all of them for the importance graph. Therefore, Figure 4.17 shows the following:

- Across all periods, these three parameters (*the mark of the assessment*, *the review patterns* and *the time to attend to review*) consistently remained the key predictors among all the classifiers.

- The feature importance converged over time. There were more diverse predictors for the first classifier. It could possibly be due to the relatively fewer items that could be reviewed and/or students were learning how to use the system.

- The parameters *review patterns* and *the time to attend to review* gradually increased their importance over time until the third exam, which highlighted the nature of programming as accumulative. Students relied on studying past assessments and attended to review them sooner.

- Another key parameter *the mark of the assessment* gained importance from the first to the second exam and maintained it from the second to the third. It become equally important how and when students had reviewed, and their previous scores in quizzes and exams.

The empirical error minimization approach was employed to determine the learning algorithm with the fewest empirical error from a bag of classifiers C [45]. Cross-validation was utilised to train and test the bag of classifiers using 10 folds. Figure 4.18 shows a visual comparative analysis between classifiers using the Receiver Operating Characteristic Area Under the Curve (ROC AUC), a well-known metric to evaluate binary classifiers, and the number for each classifier on each period is an average of the metric per fold.

In addition, Table 4.6 shows the chosen learning algorithm, SVM with a linear kernel, and the results for the weighted average precision and F1-metric, which

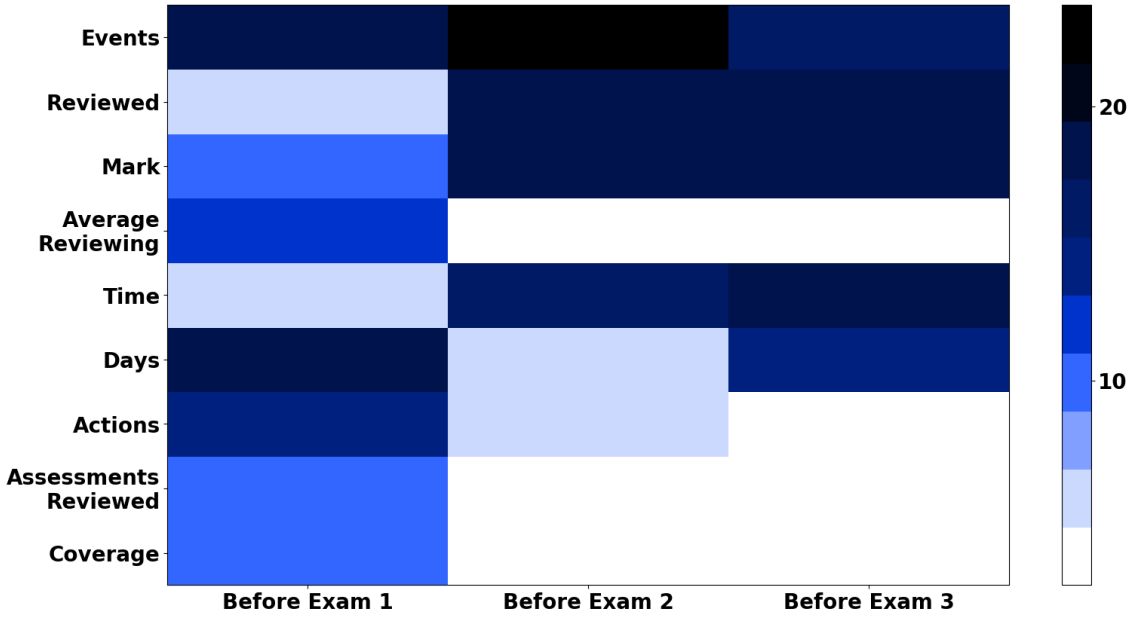


Figure 4.17: Feature Importance Across Periods for ASU's Data Structures Course

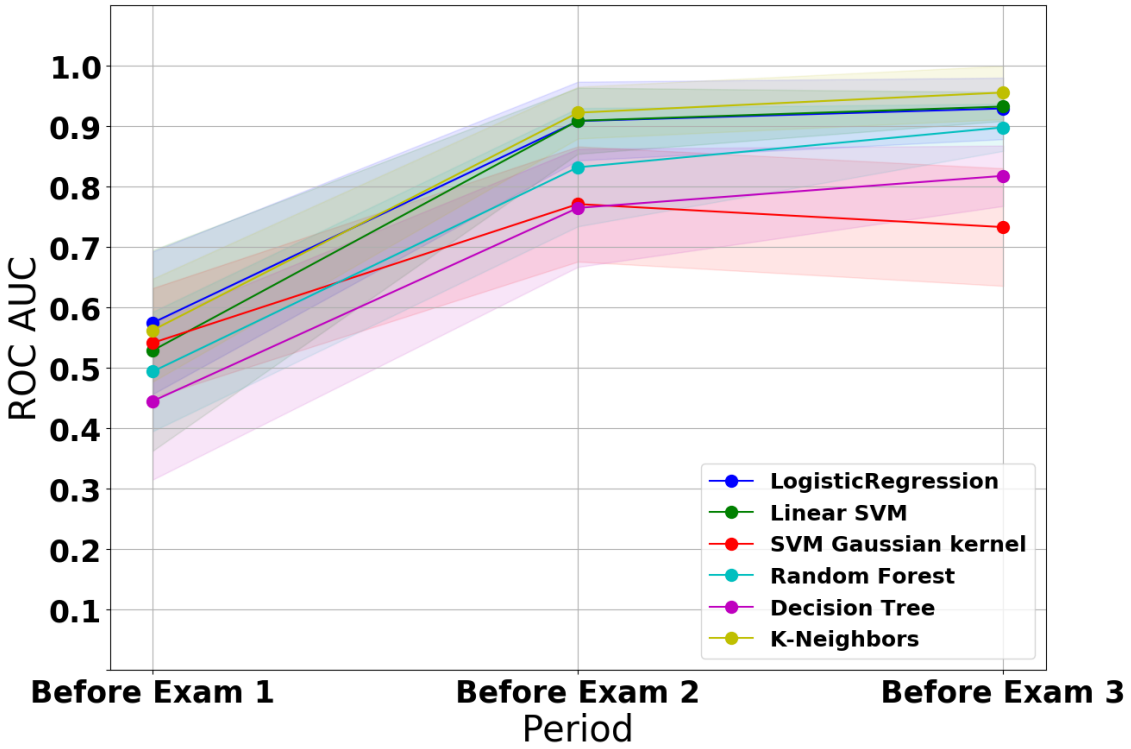


Figure 4.18: Classification Performance using ROC AUC for ASU's Data Structures Course

combines precision and recall, for each period. The values for the metrics shown are the mean and the standard deviation for the cross validation folds.

Table 4.6: Linear SVM Classification Performance throughout the periods for ASU's Data Structures Course

Period	Precision	F1-score
	Mean (SD)	*Mean (SD)*
Before Exam 1	65.05% (2.83%)	74.43% (2.27%)
Exam 1 - Exam 2	83.89% (4.58%)	86.71% (2.96%)
Exam 2 - Exam 3	90.00% (7.73%)	90.58% (6.08%)

In addition, a regression model was built to predict the precise cumulative exam average grade for each students on these periods. In a similar manner, a linear regression functions was constructed retrospectively per period. Cross-validation was employed using 10 folds. The performance of the linear regression function can be found in Figure 4.19.

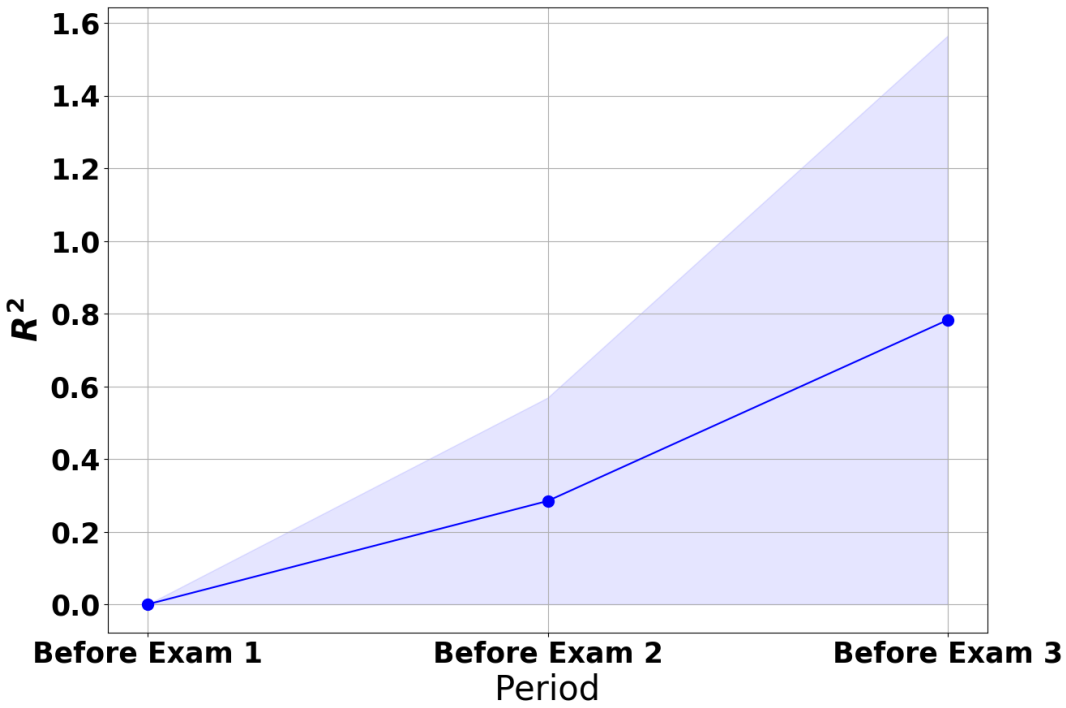


Figure 4.19: Linear Regression Performance using R^2 for ASU's Data Structures Course

Table 4.20 shows the means and the standard deviation of the folds for each

period using the Coefficient of Determination (R^2) and the Mean Absolute Error (MAE).

Table 4.7: Linear Regression Performance throughout the periods for ASU's Data Structures Course

Period	R^2	MAE
	Mean (SD)	*Mean (SD)*
Before Exam 1	0.0 (0.0)	0.0854 (0.0017)
Exam1 - Exam 2	0.28 (0.57)	0.0560 (0.0011)
Exam 1 - Exam 3	0.78 (1.56)	0.0286 (0.0057)

In Figure 4.20, the predicted cumulative target grades were plotted with respect to the actual results for each of the students before the third exam period.

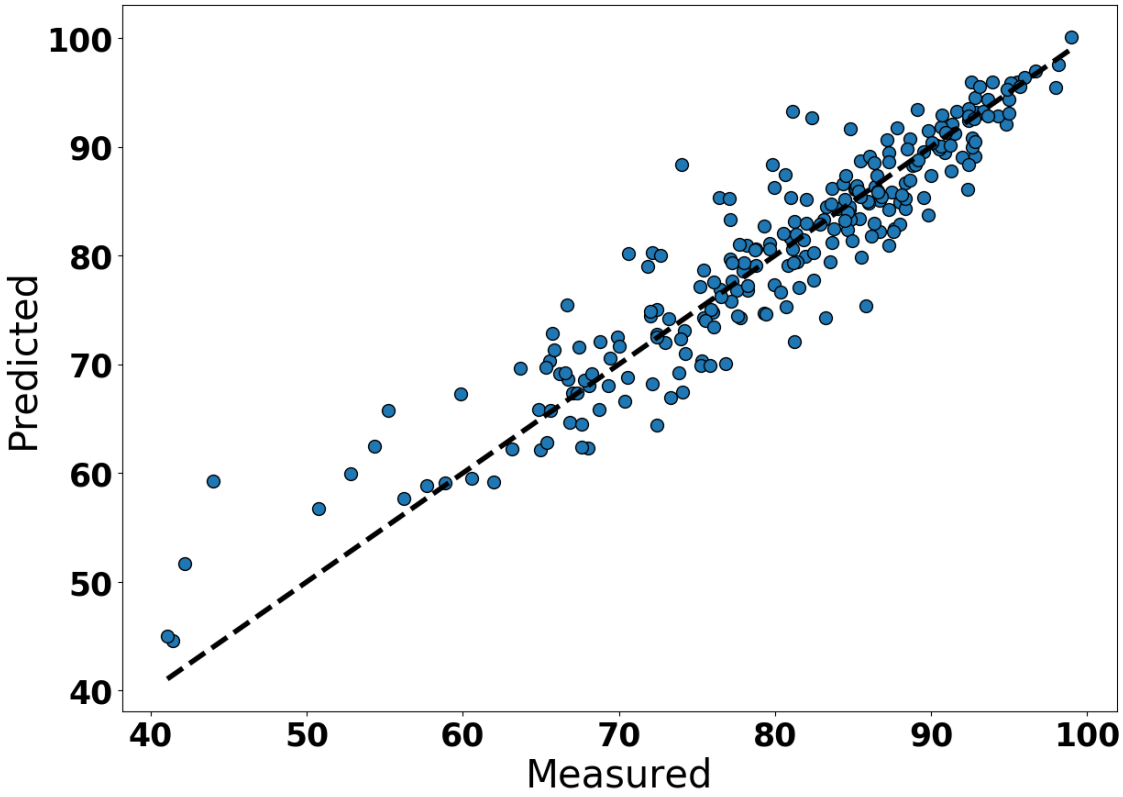


Figure 4.20: Linear Regression Predictions vs. Actual Results Before the Third Exam for ASU's Data Structures Course

4.10.4 Conclusions

This retrospective analysis managed to demonstrate:

1. the gathering of student data, which tells us their reviewing learning process
2. the extraction of features
3. the identification of patterns which could be used for predictions

RQ1 was answered, these predictions reached a usable accuracy for potential interventions and feedback to students. Both models worked well and increased their performance every week and period as students completed and reviewed more assessments and more timing and engagement features were extracted.

After the last exam was finished, the three exam scores could be utilised to calculate the average exam score and, therefore, the classification and regression models made no mistake. It is now possible to leverage the patterns extracted from this reviewing data to predict how a new incoming cohort of students will perform in next versions of this course, intervene and provide personalized help to those in need to follow desired reviewing strategies[2].

4.11 Extra: Retrospective Analysis on All First-years at DCU

As part of a DCU internal project, we were provided with a dataset of data on 16,799 first-year students over a 5-year period. Each student has up to 138 data columns (mostly categorical). This data was compiled using a variety of data sources at our university which is explained in more detail in Chapter 3.

4.11.1 Exploratory Data Analysis

We explored each of 138 parameters individually, to see how their values are distributed and to verify that the dataset given to us was valid. The number of students

[2]The code for this work can be shared on request given approval of the governing parties

whose data was made available to us, is explored as in Figure 4.21.

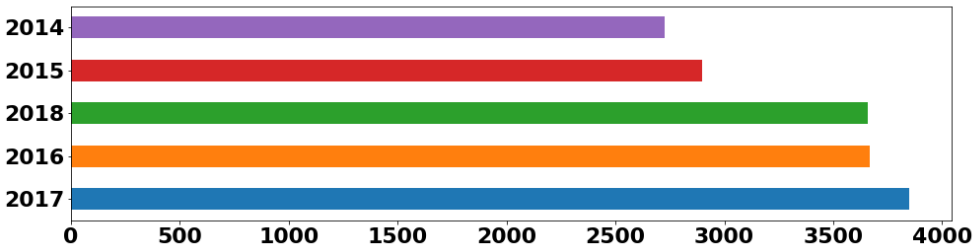


Figure 4.21: Exploring the Parameter Year from DCU's Dataset

4.11.2 Data Summarisation

As an initial exploration of this data, we measured the correlation between these parameters and each student's precision mark. The precision mark is a percentage representing the student's overall performance across the whole year. Its is computed as a weighted sum of the student's grade in each of his/her modules, the weights varying according to the number of credits awarded to each module. See Figure 4.22 for a scatter plot that shows the relationship between the CAO Points (the student's performance in the national examinations in which performance is used to select entry to University courses) and the Precision Mark colour coded by Faculty.
Figure 4.22 shows there is a correlation between CAO marks and precision mark. However, students can still do well even if they enter University with low points. If we divide this data up by Faculty, Figure 4.23 shows how there are differences in this correlation across Faculties. For instance, the Institute of Education has a small variance and most students score between 50% and 75%. This "flat" distribution may happen because the work they do in college is more important than their previous studies. In the Engineering and Computing faculty, students can perform very well and not so well even if they come with high points. However, students near the 600 points mark do excel. In Science and Health, there is a high correlation between CAO marks and precision mark but, some students still perform very well coming with low CAO points.

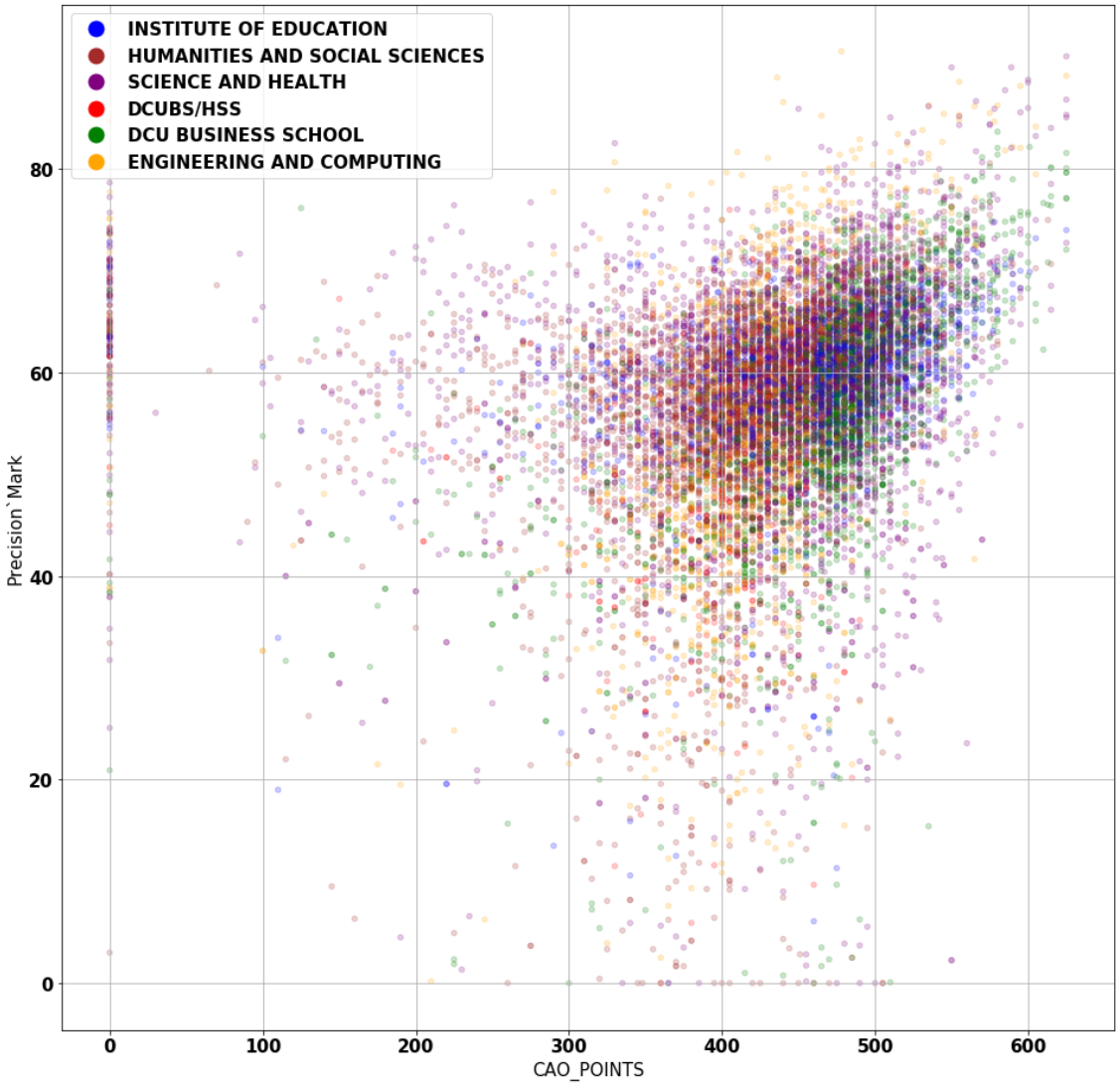


Figure 4.22: Scatter Plot between CAO Points and the Precision Mark, color coded by Faculty

4.11.3 Feature Engineering

The 138 parameters (mostly categorical) were encoded into 891 features for a ML algorithm to use this information in building a model to predict the precision mark of students. It is worth to note the resulting matrix was very sparse, with lots of non-applicable values and gaps to do with repeat students and students transferring across courses. Students were split between Training (80%) and Testing (20%) sets by keeping a balance between students that passed or failed the first assessment.

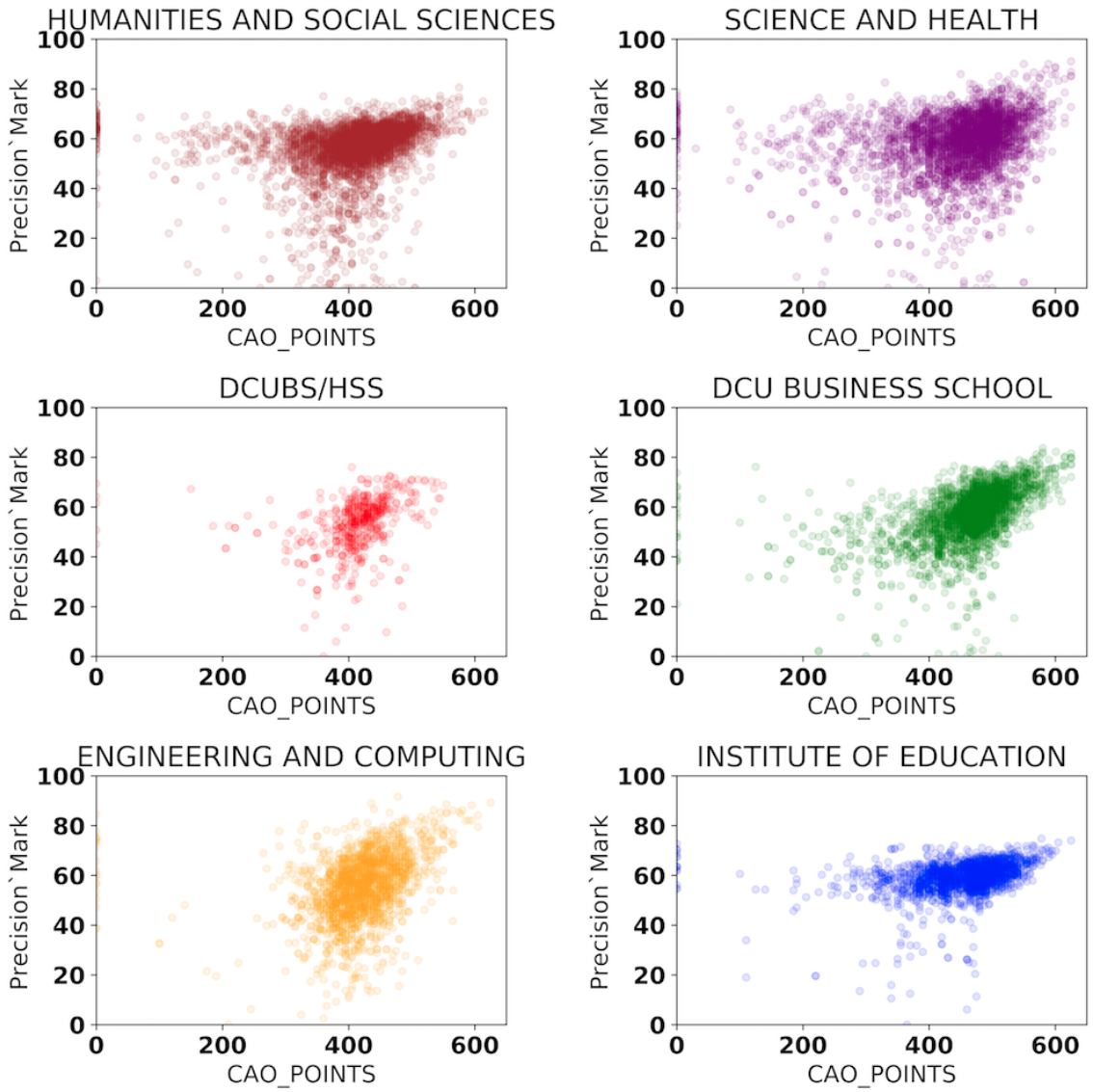


Figure 4.23: Scatter Plots between CAO Points and the Precision Mark by Faculty

4.11.4 Predictive Modelling and Ablation Studies

Training a Decision Tree or a Random Forest or any machine learning algorithm to predict which students may be "at-risk" (achieving a result below the 40% threshold of the Precision Mark) is not a difficult data analytics task. For the Random Forest we trained, the accuracy using the Testing dataset for our evaluation metric shows 95% accuracy and if we were to use the data for this we would be quite happy with this level of performance.

However, we are interested in other usages of the student data, in particular in determining the importance of different features. In order to measure how much

variation or impact each feature had on a predictive model, we leverage the concept of ablation studies. An ablation study typically refers to removing some feature of a model from the algorithm and seeing how that affects performance [65]. In our work, we excluded each feature from a linear model in order to measure how much variance the feature contain.

The result shows that the most impactful features which removed the greatest amount of variance for the model generated, are quite noisy as most of them only refer to repeat students and are blank for the rest. This is because there are a small number of repeat vs. non-repeat students so there is a bias in that aspect of the data.

4.11.5 Ablation Study & Mutual Information

An ablation study typically refers to removing some "feature" of the model developed and then measuring how that affects performance [65]. This technique has been recently used in Deep Learning Research and Computer Vision [42].

In addition, to explore the importance of the features in this student dataset, we looked at different metric, Mutual Information. The Mutual Information of two random variables measures their mutual dependence (how they vary together) [74]. Mutual Information is more general than the correlation coefficient, it can cover categorical variables and it determines how similar the joint distribution of the pair (X, Y) is to the product of the marginal distributions of X and Y. See Figure 4.24 for a number of graphical examples to illustrate what the shapes of the graphs of mutual information between two variables looks like [2].

In our work, we trained a linear model using all the features given to us from first-year students. Then, we trained models by removing features individually to understand which features impacted the most to the performance of the model individually. After discarding features that are only relevant to students that repeated the year, the most meaningful features from the ablation studies are the following:

- NUM_EARLY_LOANS: Number of early loans (to end Nov). Number of items

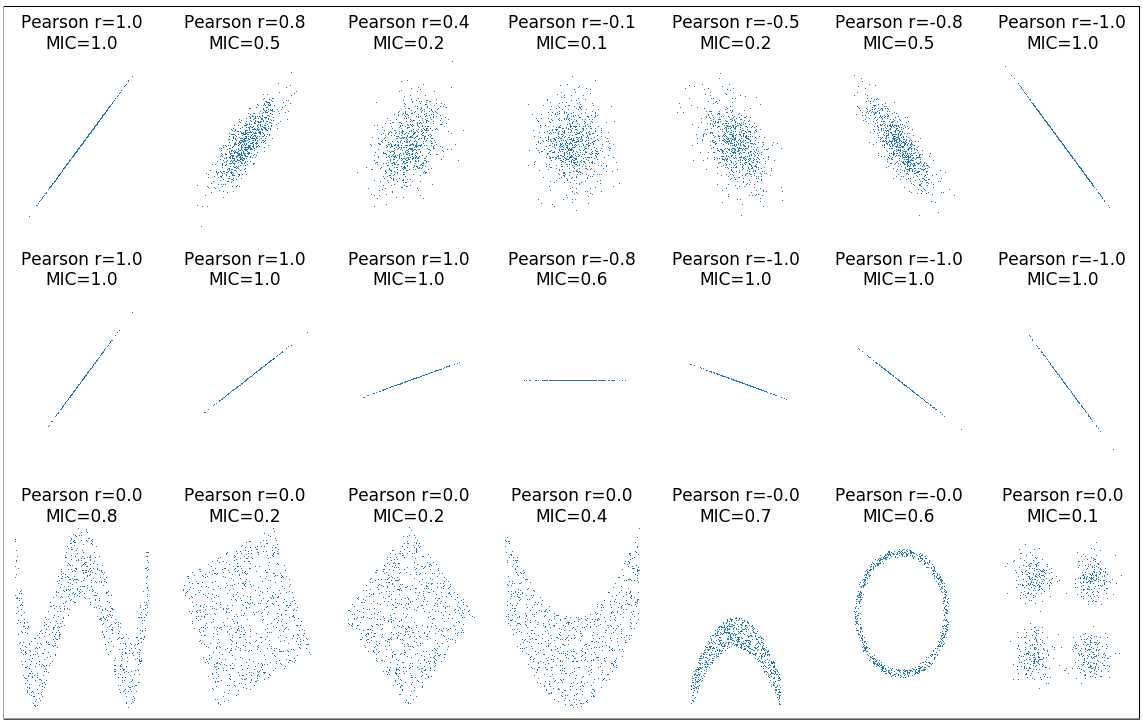


Figure 4.24: Examples of Mutual Information between Two Variables. Image taken from [2].

borrowed from the library (from September to November)

- EARLY_Q3_SUBS_PC: Number of early Third Quartile Submissions. When compared with fellow students submitting same assignment, relative quartile ranking of when assignment was submitted (irrespective of due date), due date in September, October and November
- FIRST_ASSIGN_GRADE: Max Grade for First Assignment
- NUM_GRANTS: SUSI Grants. Number of grants received in year (2015 on)
- NUM_EARLY_LATES: Number of Late Submissions September, October and November. Submissions after Due Date (only where non-ambiguous Due Date given) in September, October and November.
- NUM_SUBJECTS: Total Number of subjects taken
- NUM_MEMBERSHIPS: Total number of Club / Society memberships. Memberships as recorded for year in question

- NUM_CLUBS: Number of Club memberships
- NUM_SOCIETIES: Number of Society memberships
- NUM_M5_FAILS: Number of subjects failed in semester 2 (May). Number of failures for any fail / defer / absent reason.
- STUDENT_TYPE: Student Type, 1-9 depending on year of leaving cert and if Irish / non Irish.
- DAYS_TO_FIRST_LAB: Days to First Lab Session (full year). If null, set to 360
- CAO_POINTS: Leaving Cert points (student type 9 only)
- LEAV_CERT_MATHS: Leaving Cert Maths points. Points from best maths result (may be from resit), excludes bonus points
- YEARS_SINCE_LC: Years elapsed since Leaving Cert year

Then, we measured the mutual information between each of the meaningful features extracted from the ablation studies and the Precision Mark. The Precision Mark is an average grade from all first year courses. Figure 4.25 shows the mutual information analysis between these features and the Precision Mark. Each graph shows the Pearson correlation coefficient (and p-value) along with the Mutual Information coefficient.

4.11.6 Conclusion

Based on our work in exploring the longitudinal student data from DCU that we have access to, we could not identify a single independent feature of student data that stands above others in predicting how students are going to perform in their first-year examinations as determined by their overall Precision Mark. From this we conclude that there has to be a combination of features that will model how students behave using different data sources.

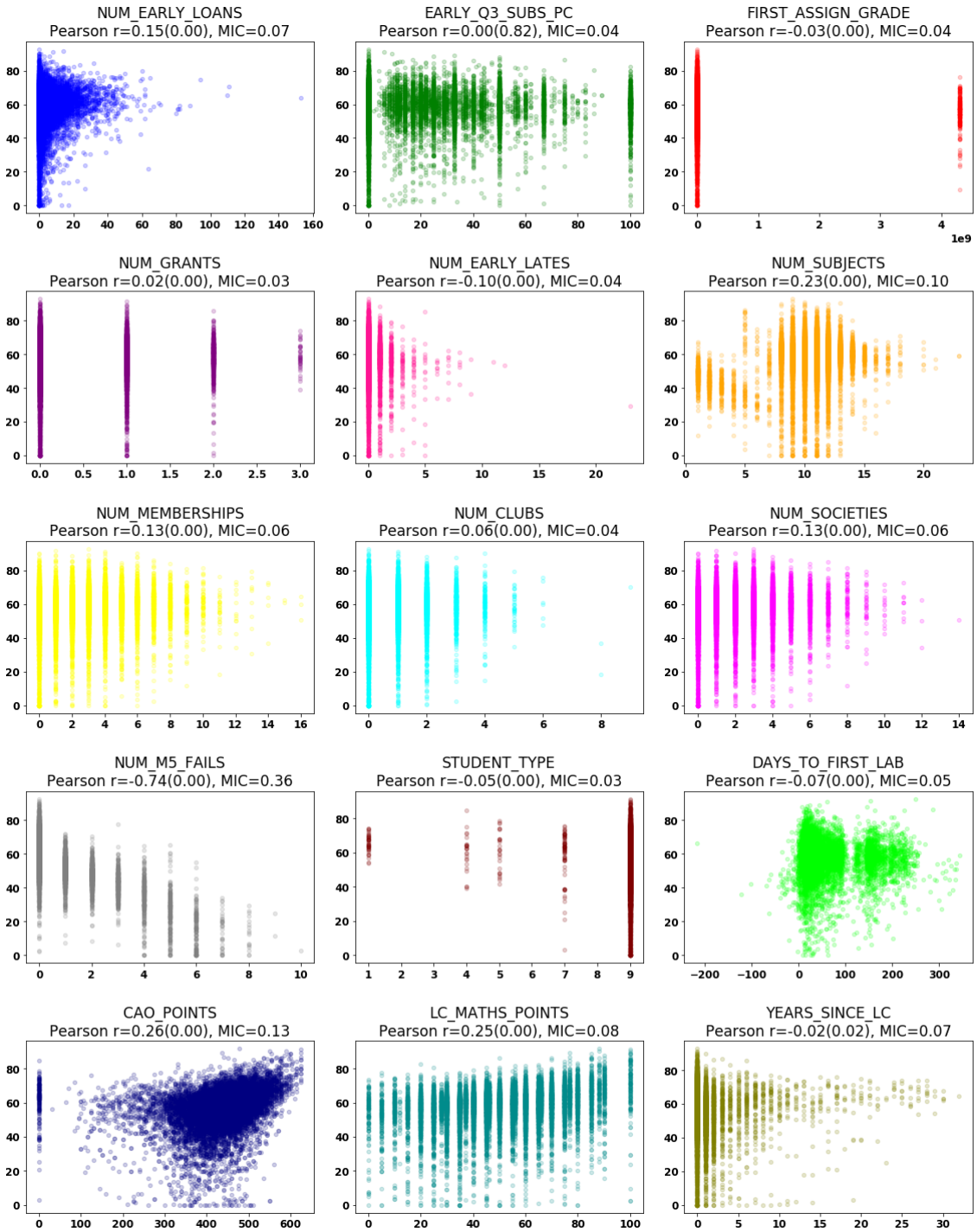


Figure 4.25: Mutual Information Score between a Feature and the Precision Mark, for Several Features

The challenge is twofold, firstly to develop more sophisticated multi-features models which would be as accurate but which would allow feature importance to be determined and this is outside the scope of this thesis and secondly to integrate the systems in the university where student data is gathered. In order to set up

a continuous stream of student data to feed systems like we have developed, a Data Lake or Data Warehouse infrastructure is needed [108]. Collecting, cleaning, crunching and aggregating student data parameters and features is viable for short-term projects or retrospective analyses. However, in order to set up production machine learning pipelines, a more time efficient infrastructure is needed. That way, our models can be re-trained in a continuous fashion, a single access to all student data will be needed and even a workflow can be developed for sending recommendations to students in an automated way.

Finally, **RQ1** was answered as true by gathering a student digital footprint of parameters combining several data sources at our university and a predictive model was built and tested retrospectively[3]. Novel techniques such as ablation studies or mutual information coefficients were used to discern meaningful parameters. However, a rich combination of the parameters should be used for representing how a student behaves.

[3]The code for this work can be shared on request given approval of the governing parties

Chapter 5

Modelling Students With Embeddings

5.1 Introduction

In this chapter, we develop a new approach to profiling and modelling individual students who are studying computer programming based on their programming design using a technique called *embeddings*. We investigate different approaches to analysing users based on their *source code* programming submissions in the Python language. We compare the performances of different source code vectorisation techniques to predict the correctness of a code submission. In addition, we propose a new mechanism to model and to represent students within our AI-based prediction system based on their code submissions for a given set of laboratory tasks, on a particular course. In this way, we can make deeper recommendations for programming solutions and personalised pathways to support student learning and progression in computer programming modules at a Higher Education Institution.

The main contributions of this chapter are as follows:

(i) Several approaches to representing students' source code submissions are investigated, describing the merits associated with each approach;

(ii) The performance of different source code representations for predicting the

correctness of student's source code are then evaluated;

(iii) A mechanism for representing student programming profiles is developed based on their vector representations and leveraging the code submission for a given course.

We begin by describing different methods for source code representation and vectorisation that were investigated. Then we discuss different ways for transforming tokenised source code into real vectors to be used for predicting the correctness of a program. This is achieved using different code representations and also our proposed method in order to represent a user or student using all their programming code submitted for exercises during a course. In effect, this is a richer representation or model of a student, being based on their actual computer coding practice. In section 5.6 we discuss the results of our experiments on predicting the correctness of a computer program. Section 5.7 presents an analysis of students for two courses based on this richer user representation, using all their program submission for a course.

5.2 Context and Dublin City University Courses

In Dublin City University, students learn how to code by taking a variety of programming modules. Students develop code algorithms for problems proposed by Faculty. Many of these courses or modules are delivered through the Virtual Learning Environment (VLE) built for the purpose of teaching and learning computer programming introduced earlier in chapter 3. This custom VLE enables students to access course information, material and slides for each module. In addition, the system integrates an automatic grading platform where students can verify their code submissions for various programming exercises. Students typically develop solutions locally for what are called *laboratory sheets* or sets of exercises and programming tasks for the computer programming courses. Then, they submit their individual programs online to the automatic grading platform which runs a number of test-

cases specified by the Lecturer on each exercise. This provides instant feedback to students based on the suite of testcases run and ultimately tells the student whether the program is considered correct or incorrect based on whether any of the testcases fail. This information is invaluable to students' learning and such a platform as this is very beneficial in order to verify the students' programs work as expected.

In this chapter we explore different mechanisms to represent students' code to predict its correctness and to better analyse students' progress in learning using their interactions. This can then be exploited to provide effective feedback and to support better personalised recommendations on further learning material to show to each student. Every time a student submits a code solution for verification, the system stores the code submission, the student identifier, the IP address of the computer used for the upload, the results of the testcases run with inputs and outputs, the course the submission belongs to, the exercise and the task name the student is attempting by using the submission's filename. In total, we collected more than half a million programming submissions (591,707) for 666 students from 5 Python programming courses over 3 academic years.

5.3 Research Method: Code Vectorisation

Machine Learning (ML) extracts patterns from data and learns rules without being explicitly programmed [17]. In Chapter 4 we introduced ML and covered how it can be used for predictive modelling. Data used by ML algorithms has to be structured information. For instance, images are processed into matrices of numbers before inputting them to a ML algorithm. However, data is rarely presented in a straightforward structured way. ML algorithms generally need to find a way to process text information like Natural Language or multimedia such as images and videos.

In order for a ML algorithm, like a Logistic Regression Model or a Support Vector Machine, to understand text, it needs to be converted into vectors of numbers. In short, text has to be encoded as numbers to be used as input or output for Machine Learning and Deep Learning models. For that, a suite of natural language processing

(NLP) techniques are employed. In our case, code submissions or programs cannot be considered as natural language and need to be parsed and analysed in a different way. We explored the following representations of programming submissions by tokenising the code:

1. Code as Word Vectors;
2. Code as Token Vectors;
3. Code as Abstract Syntax Tree Vectors.

In the following sections we dig deeper into vectorised representations within a mathematical model based on using student programming code. For that, we support our narrative with Listings 5.1, 5.2, and 5.3. These examples are code snippets similar to students' submissions in our programming courses.

Listing 5.1: Hello World Example

```
#!/usr/bin/env python
print "Hello, World!"
```

Listing 5.2: Call a Function Example

```
#!/usr/bin/env python
def say_hello():
    print("Hello, World!")
say_hello()
```

Listing 5.3: Sum of Two Variables Example

```
#!/usr/bin/env python
# read from input
a = int(raw_input()) # first
b = int(raw_input()) # second
print a + b
```

5.3.1 Program Code as Word Vectors

A straightforward and simple approach to representing computer programs as objects in a machine learning model is to leverage the words from the code solutions as input into a machine learning algorithm. For each programming submission, we split the submission using only the space, tabular (`\t`) and new line (`\n`) characters. A typical tokeniser uses characters such as exclamation marks and other operands and operators. In a coding scenario, these characters play a key role in code submissions and we do not use them as filters for our tokeniser.

Listings 5.4, 5.5 and 5.6 show how such word vectors are extracted, prepared and made ready to use for some of our snippets.

Listing 5.4: Array of Words for Hello World Example

```
['print', '"hello,', 'world"']
```

Listing 5.5: Array of Words for Call a Function Example

```
['def', 'say_hello():', 'print("Hello,',
 'World!")', 'say_hello()']
```

Listing 5.6: Array of Words for Sum of Two Variables Example

```
['a', '=', 'int(raw_input())',
 'b', '=', 'int(raw_input())',
 'print', 'a', '+', 'b']
```

These word vectors may not represent a programming submission in a very comparable way to other submissions that have, for instance, different variable names. Even though the special characters like operands carry important information regarding these code programs, splitting the words only using spaces may not give a useful representation.

5.3.2 Program Code as Token Vectors

As we are working with the Python programming language, we leverage Python's own Tokeniser[1] library for source code analysis. This module provides a lexical scanner for Python source code and is itself also implemented in Python. For instance, Listings 5.7 and 5.8 show operators and delimiter tokens are clearly identified and are assigned the generic OP token category in our example code snippets. We hypothesise this fine-grained tokenisation such as the generalisation of operands (OP), strings or names can help determine a more representable vectorisation of a code submission.

Listing 5.7: Token Categories for Hello World Example

Characters	Category	Token
1,0−1,5:	NAME	'print'
1,6−1,20:	STRING	'"Hello, World"'
2,0−2,0:	ENDMARKER	''

Listing 5.8: Token Categories for Call a Function Example

Characters	Category	Token
1,0−1,3:	NAME	'def'
1,4−1,13:	NAME	'say_hello'
1,13−1,14:	OP	'('
1,14−1,15:	OP	')'
1,15−1,16:	OP	':'
1,16−1,17:	NEWLINE	'\n'
2,0−2,4:	INDENT	'␣␣␣␣'
2,4−2,9:	NAME	'print'
2,9−2,10:	OP	'('
2,10−2,25:	STRING	'"Hello, World!"'
2,25−2,26:	OP	')'

[1]For Python 3: https://docs.python.org/3/library/tokenize.html

2,26−2,27:	NEWLINE	'\n'
3,0−3,1:	NL	'\n'
4,0−4,0:	DEDENT	''
4,0−4,9:	NAME	'say_hello'
4,9−4,10:	OP	'('
4,10−4,11:	OP	')'
4,11−4,12:	NEWLINE	'\n'
5,0−5,0:	ENDMARKER	''

These token categories or types have an associated identifier that can also be used for vectorisation, as shown in Listing 5.9.

Listing 5.9: Token IDs for Call a Function Example

Characters	Category	Token
1,0−1,3:	1	'def'
1,4−1,13:	1	'say_hello'
51,13−1,14:	51	'('
51,14−1,15:	51	')'
51,15−1,16:	51	':'
4,16−1,17:	4	'\n'
5,0−2,4:	5	'␣␣␣␣'
1,4−2,9:	1	'print'
51,9−2,10:	51	'('
3,10−2,25:	3	'"Hello,␣World!"'
51,25−2,26:	51	')'
4,26−2,27:	4	'\n'
54,0−3,1:	54	'\n'
6,0−4,0:	6	''
1,0−4,9:	1	'say_hello'
51,9−4,10:	51	'('
51,10−4,11:	51	')'

0 ,0 −5 ,0: 0 ’ ’

Although these tokens appear to represent code solutions more meaningfully than word vectors do, information regarding the structure, design and flow of the program is still not captured and to do this requires an even more complex representation, as we shall see in the next sub-section.

5.3.3 Program Code as Abstract Syntax Tree Vectors

In order to preserve the structure of the source code in a student submission, we also analyse the code submissions using Abstract Syntax Trees (ASTs). An AST is a tree representation of the abstract syntactic structure of source code, independent of what programming language the code is written [77]. An AST is an abstract representation as there are no details regarding the correctness of the implementation but only the structure and content of the code. For instance, operands are implicit in the AST and IF or While expressions are denoted with a tree node. Figures 5.1, 5.2 and 5.3 are custom visualisations[2] after recursively traversing the nodes from the AST trees generated from our example code snippets. Green nodes represent terminal nodes or leaves. Nodes that have children are coloured in blue.

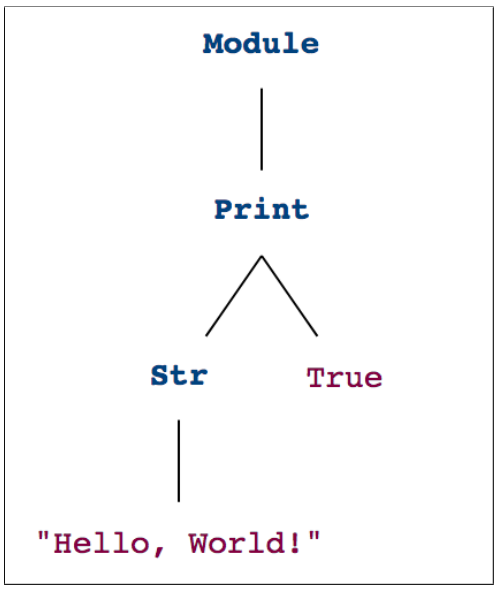


Figure 5.1: Abstract Syntax Tree (AST) for Hello World Example

[2] https://github.com/hchasestevens/show_ast

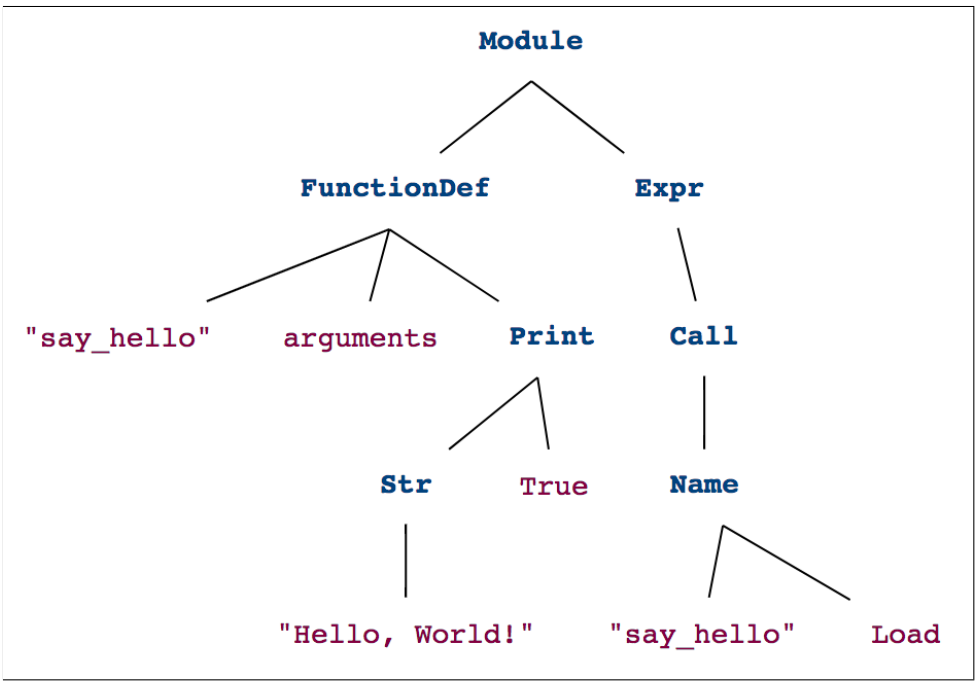


Figure 5.2: AST for Call a Function Example

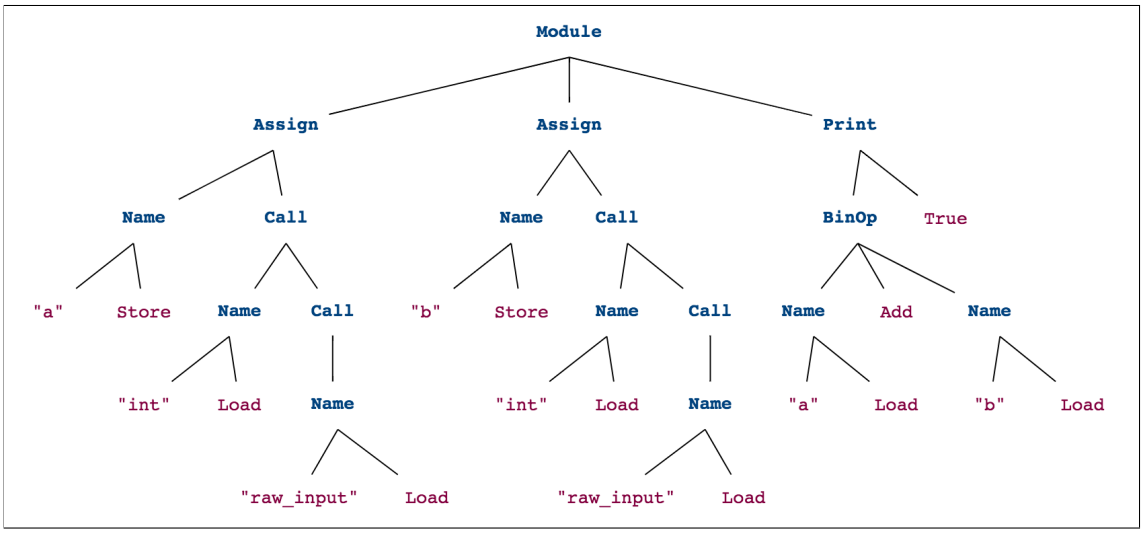


Figure 5.3: AST for Sum of Two Variables Example

After traversing the ASTs, nodes can be represented using their parents in a pair-wise way. See Listings 5.10 and 5.11 for two of the example snippets. The ASTs are traversed using a Breadth-first search (BFS) approach.

Listing 5.10: AST Pairs for "Hello World" Code Example

```
Parent Node      Child Node
'Module'         'Print'
```

```
'Print'          'Str'
'Print'          'True'
'Str'            'Hello\tWorld!'
```

Listing 5.11: AST Pairs for "Call a Function" Code Example

```
Parent Node      Child Node
'Module'         'FunctionDef'
'Module'         'Expr'
'FunctionDef'    'say_hello'
'FunctionDef'    'arguments'
'FunctionDef'    'Print'
'Expr'           'Call'
'Print'          'Str'
'Print'          'bool'
'Call'           'Name'
'Str'            'Hello\tWorld!'
'Name'           'say_hello'
'Name'           'Load'
```

5.4 Experiment: code2vec

We now investigate how student code submissions can be transformed into meaningful vectors as a form of representation of the program code, and implicitly as a representation of the student who submitted that code. As mentioned earlier, computers do not understand text data and text needs to be represented and encoded into vectors of numbers as the input into a Machine Learning algorithm. For that, we use the following two approaches:

1. Code BOW (bag-of-words)
2. Code Embeddings

The number of words extracted after running the tokeniser on our data are 231,659 which was fitted with 591,707 code submissions. A lot of computer memory is required to generate the large sparse matrices for learning code2vec and user2code2vec representations. Although our experiments are run on a GPU for faster computation, running a classification algorithm for more than half a million source code files is computationally expensive, hence we set a limit to the number of Words, Python Categories, Python Tokens Words and AST Nodes to 2,000. Overall, there are fewer Token Words than Words.

5.4.1 Code BOW (bag-of-words)

The bag-of-words (BOW) model, also called the vector space model, is a simple representation of documents and queries used Information Retrieval for almost 50 years [94]. According to this model, a text (such as a sentence or a whole document or a user query) is represented as a bag of its words, disregarding grammar and even word order but keeping multiplicity. In our work, we leverage the BOW model to represent code submissions by looking at either:

(a) Words

(b) Python Token Categories

(c) Python Token Words

(d) AST Nodes

The ordering of these items in each of the alternatives is ignored and only their frequency is stored in a large sparse matrix. This matrix can be populated using one of the following operations:

- Count: count of each word in the document.
- Frequency: frequency of each word as a ratio of words within each document.
- Binary: presence, whether or not each word is present in the document.

- TF-IDF: Term Frequency times Inverse Document Frequency (TF-IDF) scoring for each word in the document [94].

Table 5.1 shows a simple BOW example using the count of each Token Word for Listings 5.1 and 5.2 as the corpus. This BOW approach can be used for classification methods where the count, frequency, presence or TF-IDF of occurrence of each item (Word, Token Category, Token Word or AST Node) is used as a feature for training a classifier.

Table 5.1: Count Occurrence Matrix for Listings 5.1 and 5.2

UNK	'('	')'	'print'	'"Hello, World!"'	'say_hello'	'def'	':'
0	1	1	1	1	0	0	0
0	3	3	1	1	2	1	1

5.4.2 Code Embeddings

The BOW model provides an order-independent representation of computer program source code, where only the counts of either (a) Words, (b) Python Categories, (c) Python Tokens Words or (d) AST Nodes matter. In contrast, embeddings are a different type of feature learning technique in NLP where items, words (or even phrases) from the vocabulary are mapped to vectors of real numbers [66]. It involves a mathematical embedding from a space with one dimension per word to a continuous vector space with a much lower dimensionality.

We generate embeddings for code submissions from our students by transforming them into vectors in a continuous vector space. In a similar manner, we leverage the vectorisation of the code solutions proposed in Section 5.3. We hypothesise that embeddings extract patterns using contextual information and when used in combination with a Neural Network can predict the correctness of the code solutions more effectively. Embeddings typically uncover really interesting properties or relationships between items or words such as neighbourhoods of items or classes,

relationships between items or constant vector differences, which we describe in the next section.

5.5 Experiment: user2code2vec

user2code2vec is a novel idea and mechanism for representing student programming profiles developed entirely by us. It is based on vector representations leveraging the code submissions from students for a given course.

Students typically submit programming versions of the same exercise proposed in the labsheets to the grading platform until they either get it correct, or they give up. There is no limit on the number of student submissions per exercise. Then, for each user and each proposed exercise or task, the grading platform contains a set of versions. In our work, we only leverage the latest version per task for the vectorised representation of the user. In a similar manner, if we wanted to keep all versions, we would add another dimension and develop a tensor with all the submissions.

For each course and academic year, a **User Representation Matrix** is constructed for each student using the code vectors of the submissions to the proposed labsheets by the Lecturer. Having a vector representation of code submissions allows researchers to generate a higher-level representation for each student or user. This User Representation Matrix is built by vectorising the submissions. Submission are vectorised using either:

1. Word Tokeniser

2. Token Word Python Tokeniser

This results in a User Representation Matrix of shape (number_tasks, MAX_LENGTH). MAX_LENGTH is the limit for each sequence that we use for padding the code submission after tokenisation. MAX_LENGTH is set to 50. The User Representation Matrix for each student is flattened out as a long vector. Principal Component Analysis (PCA) [101] is leveraged as the dimensionality reduction technique to vi-

sualise the 100-dimension vectors or user embeddings into 2 dimensions. In short, a student is represented as a vector of her submissions.

5.6 RQ2 Results: code2vec

In this section, the results of the code2vec technique will be discussed for both approaches: BOW and Embeddings. We train the models and learn representations using all the Python programs submitted by students from previous cohorts in our University over a number of years, and we use these representations to predict the correctness of code submitted by a student from the present cohort. As a reminder to the reader, in section 1.1 the second research question in the thesis was introduced and it is re-stated below:

RQ2: How can students' programming submissions be encoded into vectors for use as internal representations of those students?

This will help the reader to understand the context of the experiments we now present.

5.6.1 Code BOW (bag-of-words)

First, we build four tokenisers constructed and fitted with the code submissions using either (a) Words, (b) Python Categories, (c) Python Token Words or (d) AST Nodes, respectively. The dictionary of items and their counts are shown in Tables 5.2 and 5.3. It is interesting to see the differences between the top occurrences for each tokenisation, where Token Words are a generalisation of Words, Token Categories are a generalisation of Token Words and the AST nodes are at an abstract level which contain items regarding the structure of the code submission.

By processing student data in this way, we can construct matrices where each row is a code submission and we count the number of occurrences for each (a) Word, (b) Token Category, (c) Token Word and (d) AST Node. Figure 5.4a shows details

Table 5.2: Top-5 Words & Token Categories in terms of Number of Occurrences

Word	Occurrences	Token Category	Occurrences
'='	2,440,154	51: 'OP'	20,368,593
'i'	910,221	1: 'NAME'	18,075,194
'+'	575,607	4: 'NEWLINE'	5,886,806
'if'	552,539	2: 'NUMBER'	2,317,086
'def'	522,536	54: 'NL'	1,996,531

Table 5.3: Top-5 Token Words & AST Nodes in terms of Number of Occurrences

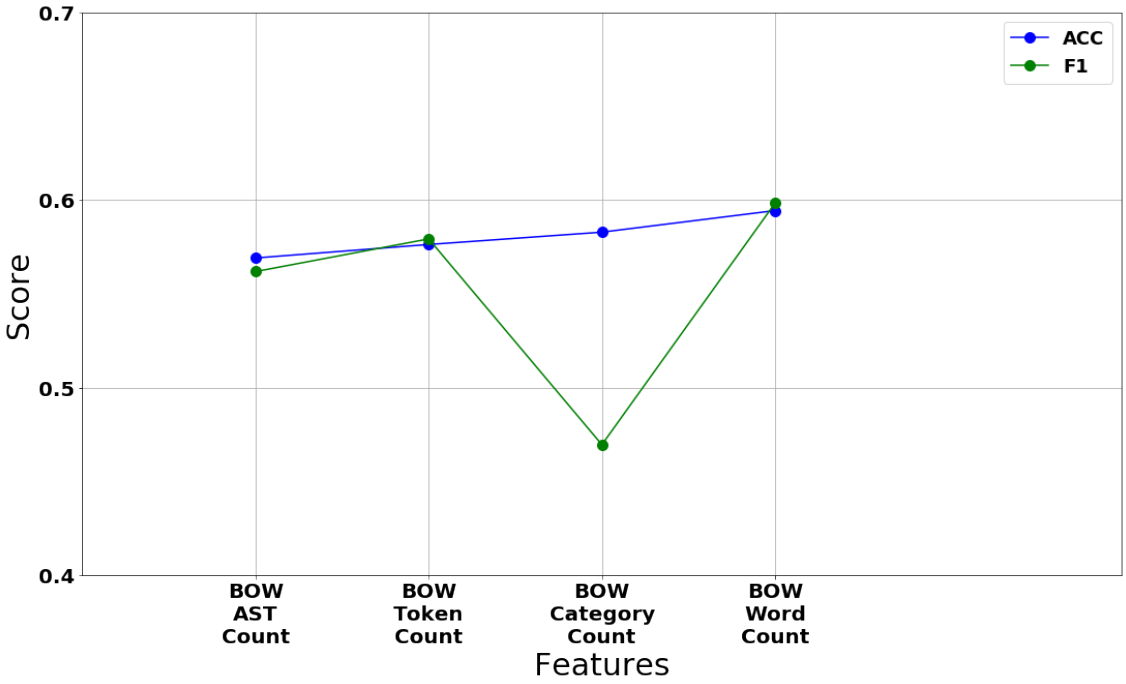
Token Word	Occurrences	AST Nodes	Occurrences
')'	3,556,931	'Name'	10,005,368
'('	3,556,907	'Load'	9,607,682
'='	2,581,991	'Store'	2,665,169
':'	2,248,901	'Call'	2,205,672
'.'	2,011,442	'Assign'	2,186,523

of the performance of these model combinations (a), (b), (c) and (d) just using the count of items. In addition, we look on the (a) Words (as they work better) and perform a similar analysis looking at the count, presence (binary), frequency and TF-IDF of the Words instead of the pure count only. Figure 5.4b does not show a meaningful difference between them except that the frequency model works slightly worse than the others. These models are trained using a Naive Bayes classification algorithm [63] holding out 20% of the data as the testset. The models are trained using around half a million code submissions (less for the Tokens or AST Trees as some code submissions could not be tokenised using the Python Tokeniser library or an AST could not be extracted when the programs are incorrectly constructed). The classes for this classification problem are well balanced. For instance, for the model that uses the Words, 194,451 submissions were correct and 296,369 were incorrect based on the output of the grading platform. That is the target of our predictions for training the models.

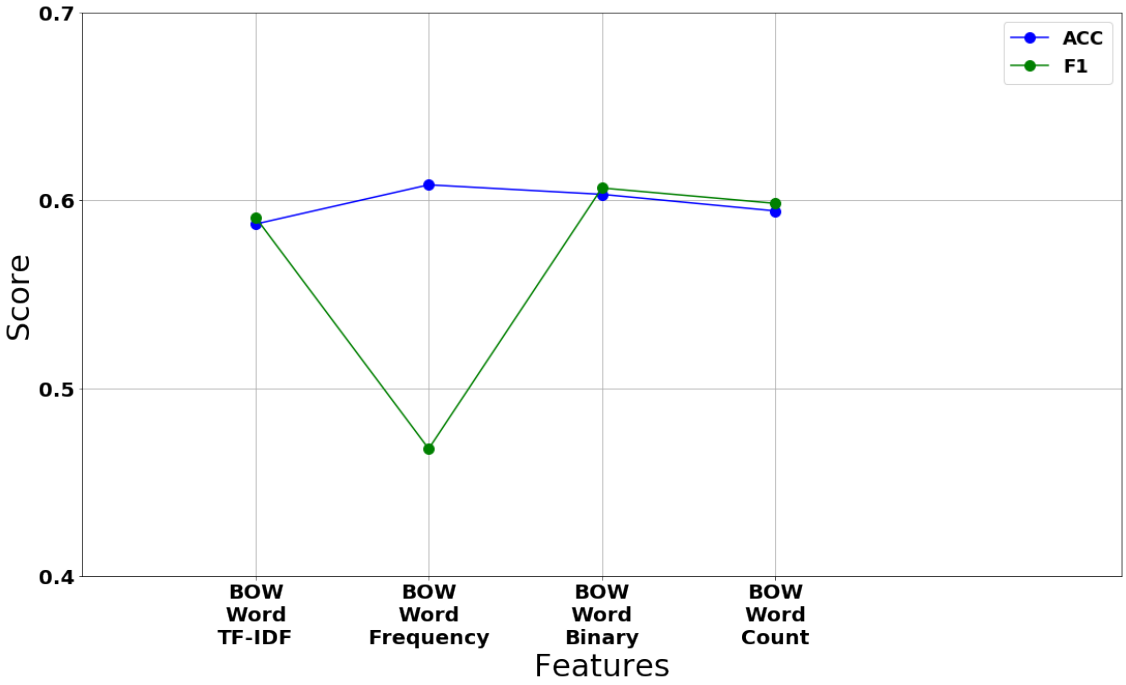
Interestingly, the least generalised model that uses the Words instead of Tokens or AST Nodes is the one that performs slightly better than the rest using BOW. The less generalised the model is, the better it performs, and using Words performs better than Tokens and Tokens perform better than AST Nodes.

5.6.2 Code Embeddings

In a similar manner and in order to feed vectors to a Neural Network, we vectorise our code submissions by tokenising for (a) Words, (b) Python Categories, (c) Python Token Words and (d) AST Nodes. In addition, as a pre-processing step, we pad our sequences up to our limit of 50 words, tokens or nodes. A simple model is developed using an embeddings layer, flattening the output of that layer on the next one and condensing it on the final one using a softmax function. The embeddings are the representation extracted after learning from the embeddings layer and contain 100 dimensions. This forces the Neural Network to learn patterns as we are inputting 2,000 words that will be 2,000 dimensions using one-shot encoding.



(a) Words vs. Token Categories vs. Token Words vs. AST Nodes Using Count



(b) Words Using Count, Binary, Frequency & TF-IDF

Figure 5.4: Performance of code2vec using BOW (bag-of-words).

The performance of the models using (a) Words and (b) Token Words is shown Figure 5.5. These models are trained using Cross Validation with 20% of the dataset as the holdout set. Utilising Neural Networks with an embeddings layer allows us to learn better patterns and representations of the code solutions submitted to the

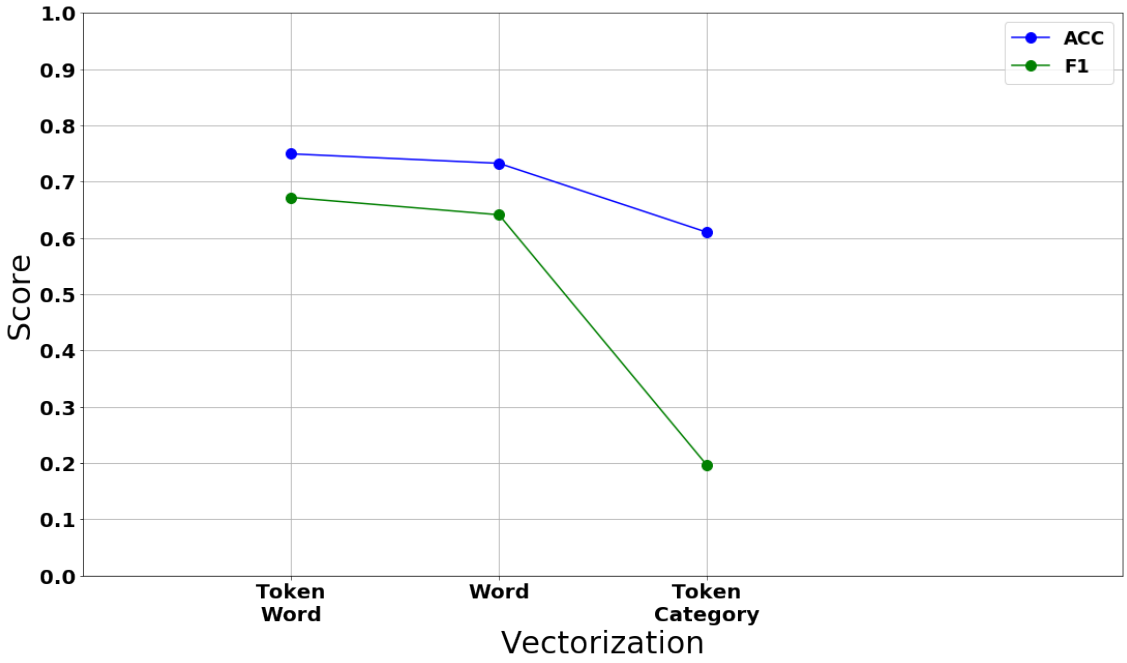


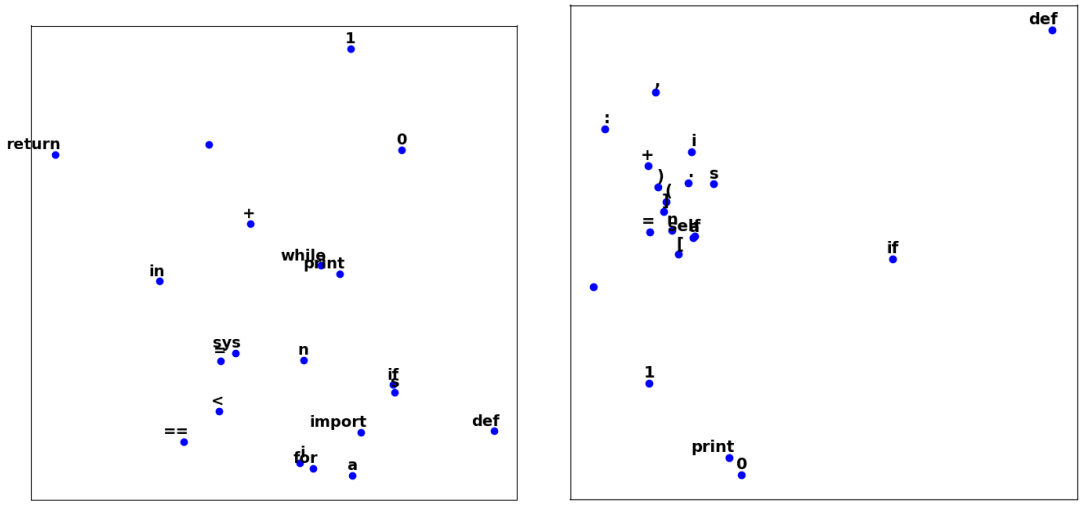
Figure 5.5: Performance of code2vec using Embeddings

grading platform. The models perform better than the baseline BOW and the Word Tokens are better able to distinguish between correct and incorrect programs. We expect that incorporating the structure of the program using ASTs will create a richer model.

Table 5.4 shows the results from the different BOW and embeddings models in a comparable way.

Table 5.4: Performance of the Models Using BOW and Embeddings

Model	Accuracy	F1 Score
Naive Bayes using BOW & Words	59.44%	59.84%
Naive Bayes using BOW & Category Tokens	58.29%	46.95%
Naive Bayes using BOW & Word Tokens	57.63%	57.93%
Naive Bayes using BOW & AST Nodes	56.91%	56.18%
Neural Network using Embeddings & Words	73.25%	64.11%
Neural Network using Embeddings & Category Tokens	74.93%	19.64%
Neural Network using Embeddings & Word Tokens	74.93%	67.18%



(a) Embeddings for the Top 20 Most Common Words

(b) Embeddings for the Top 20 Most Common Token Words

Figure 5.6: Embeddings for the Top Words & Token Words. These Embeddings Are Projected from 100 Dimensions to 2 Dimensions for Visualization Using Principal Component Analysis (PCA). Axis in the Graphs Are the PCA's Two Principal Components.

After the models are trained using the code submissions with the correct or incorrect target for each, the learned embeddings can be extracted. Figure 5.6a shows the embeddings of the top 20 most common words. It is interesting to note how operands are clustered together as are numbers. This confirms that the network is learning efficient representations. Similarly, we can explore the top 20 most common tokens in Figure 5.6b.

These vectors contain really interesting properties similar to word embeddings. Table 5.5 shows some cosine similarities between pairs of words that are very close to other pairs. Neighbors of these embeddings can also be checked out, and numbers can be found besides other numbers in String format, but in general, our learned embeddings have noise such as variable names and Strings that prevents us to from seeing other relationships as would be found in word2vec [67].

RQ2 is answered by implementing code2vec. We encoded the programming submissions using either bag-of-words or embeddings. The power of embeddings are proven with respect to bag-of-words based representations of source code. In addition, embeddings contain interesting properties that enable us to compare between

Table 5.5: Cosine Distance Between Word Vectors

$Token_i$	$Token_j$	Cosine Distance
'('	')'	0.9136
'¡'	'¿'	0.9241
'['	']'	0.9792
'if'	'elif'	0.9732
'}'	']'	0.8857
'+'	'-'	0.8846

items.

5.7 RQ3 Results: user2code2vec

user2code2vec has been performed on students of two of the computer programming courses at Dublin City University for a full academic year. Course details can be found in Table 5.6. To remind the reader, in section 1.1 the third research question was introduced and it is re-stated below for convenience:

RQ3: By leveraging the vectorisation of code submissions for a given course, how can we represent students based on their programming work?

User Representation Matrices were constructed using the code submissions for each student. Then, we flattened them to input them to a Deep Learning Network similar to code2vec with an embeddings layer that learns representations of users in a continuous space with reduced dimensionality. User embeddings are given 100 dimensions. The input data are very large and sparse vectors with the indexes of the vocabularies for the code submissions.

In CA116 (Computer Programming) during 2016/2017, these student vectors have 13,800 dimensions as there are 276 tasks to be completed in the course and the limit of the submission sequences is 50. Figure 5.7a shows the input to the Neural

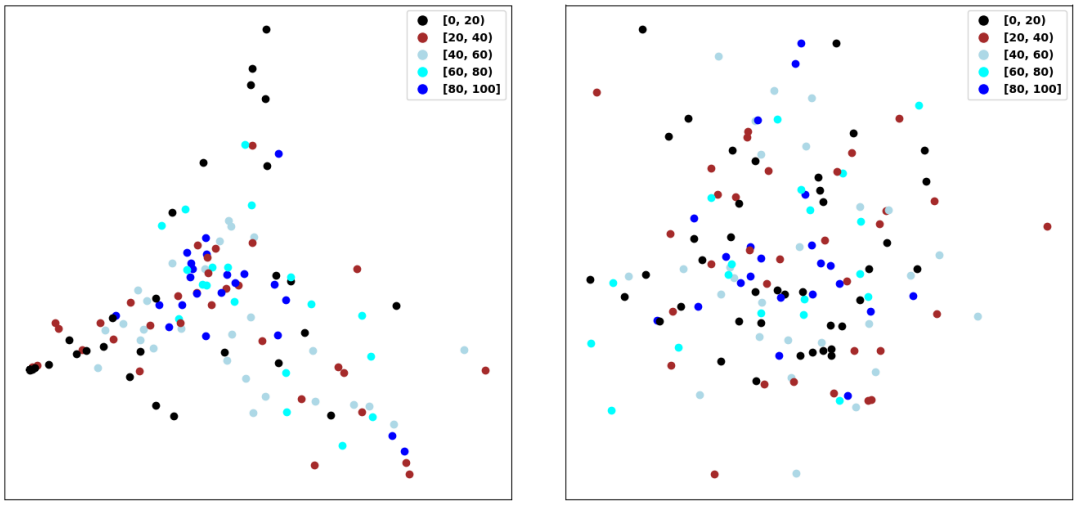
Network and Figure 5.7b shows how difficult it is to distinguish among a few hundred students with such a large sparse matrix of code submissions. Unfortunately, we cannot add more data as there were no more students in that cohort, unlike other domains which allow downloading of more tweets or crawling more websites when a similar situation with insufficient data occurs. The vectors are transformed to 2 dimensions using PCA. The variance retained is very low (between 2% and 6%). Each dot in the graphs represents a student based on the projection of their student vector. The colour used represents the average grade of the exams that student took that year on that course[3].

Table 5.6: Courses Analysed on user2code2vec

Course	Year	Code Submissions	Tasks	Students
CA116	2016/2017	68,313	276	126
CA117	2016/2017	74,065	132	140

In this section, we answered **RQ3** and proved a student can be represented as a large vector of her submissions. However, profiling students based on this information is a much more challenging task which requires more data. Deep Learning is known to work well when more training data is available and it is expected that more data will result in improved performance across most domains. Due to the curse of dimensionality, in a high-dimensional feature space with each feature having a range of possible values, typically an enormous amount of training data is required to ensure that there are several samples with each combination of values. A typical rule of thumb is that there should be at least 5 training examples for each dimension in the representation [100]. However, the constraint for Learning Analytics in Education using VLEs, but not necessarily in MOOCs, is the number of students enrolled in a course. Thus instead of representing each student using the concatenation of all their submission made in a course, it would be better to identify important features

[3]Code developed in this thesis has been made publicly available as a repository on Github at https://github.com/dazcona/user2code2vec where further details such as PCA graphs for the learned embeddings can be found.



(a) User Raw Representations Using Word Tokens

(b) User Learned Embeddings Using Word Tokens

Figure 5.7: user2code2vec applied to CA116 course during 2016/2017 academic year. These Representations Are Projected from 100 Dimensions to 2 Dimensions for Visualization Using Principal Component Analysis (PCA). Axis in the Graphs Are the PCA's Two Principal Components.

from each submission and to concatenate key features across the code submissions. In short, the approach is to keep the number of features small in order to effectively learn from constrained data. These user2code2vec representations can then be used to identify student neighbours for programming recommendations.

Chapter 6

Adaptive Feedback to Students

6.1 Introduction

This chapter builds on the work developed in chapter 4 where we built models to distinguish higher-performing students from lower performers, in semester examinations. Students who are struggling with understanding course material may be doing so for a variety of reasons and each student may not have the same issues in understanding, as the others. In order to personalise the way our students learn programming skills and to support adaptive feedback in the computer programming modules, we started sending customised weekly notifications via email. We provided this feedback typically after week six (mid-semester) of the teaching period. This chapter then presents a study on students' engagement with the weekly personalised performance notifications. Overall, the predictive and personalised feedback helped to reduce the gap between the lower and higher-performing students. Furthermore, students praised the prediction and the personalised feedback, conveying strong recommendations for future students to use the system. We also found that students who followed their personalised guidance and recommendations performed better in subsequent examinations.

6.2 Learning Context

Students have different needs for support at different times during their learning periods. Understanding which students are finding difficulties with course material at different stages is a potentially great resource to help improve learning and ultimately success in passing a course. However, students are likely to have different knowledge gaps from one another (Figure 6.1).

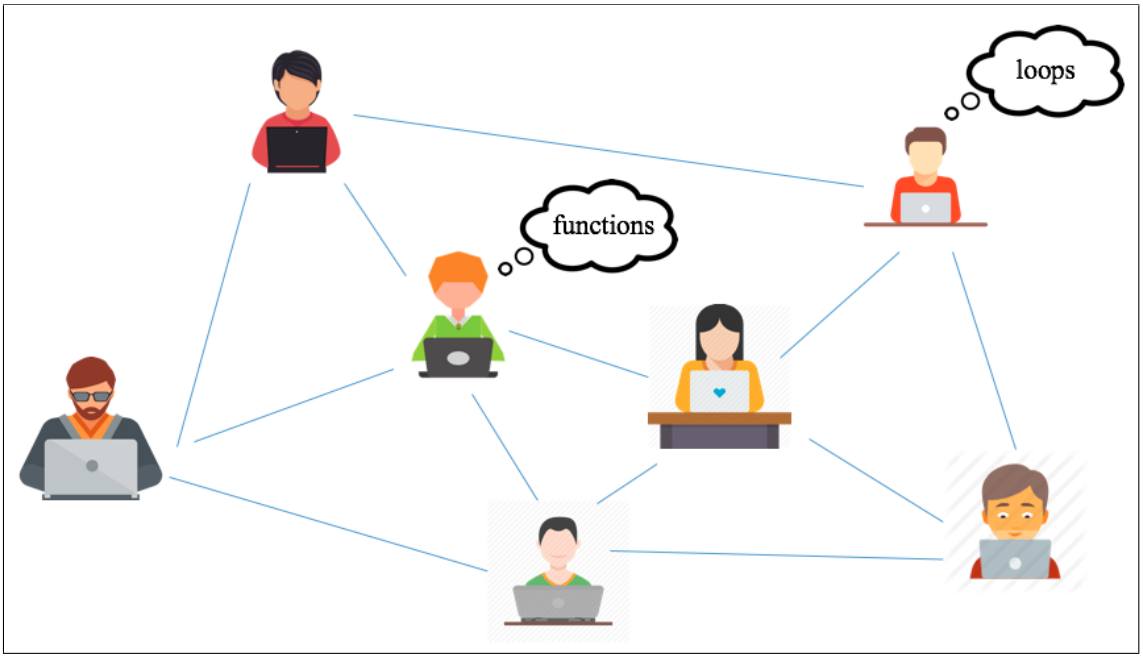


Figure 6.1: Students Struggling with Programming Concepts May Have Different Learning Issues

As mentioned earlier, in Chapter 3, computer programming modules in our institution are delivered through the custom VLE platform which allows students to access their learning material online. In addition, students are able submit and to verify the correctness of their computer programming work in real-time on the VLE where a suite of unit test-cases are run on their assignment submissions. The students' digital footprint gathered can then be leveraged using Artificial Intelligence and Machine Learning techniques and combining them with other student data information to identify students having issues, as described in our published work [10].

In order to adapt to students' learning on this VLE platform, in the middle of the semester, a feature is enabled on the VLE for students to opt-in or opt-out of receiving weekly personalised notifications on their progress, relative to other students. These include a performance message based on the predictions being run on the class of students to which they belong which has been trained with a combination of historical student cohorts data, recommended learning material and laboratory sheets resources to review based on their progress and finally programming code solutions from among the top-ranked students in their class as well as additional support resources.

We describe the details of this in this chapter.

6.3 Research Methodology

6.3.1 Feedback to Students

Feedback was sent to students who decided to opt-in to receive these feedback notifications via email. The feedback to each student was personalised in the following ways:

- By leveraging our weekly predictions and based on the associated probability of failing the next laboratory exam, students were ranked and divided into deciles. Hence, there were 10 custom messages we sent based on their performance. From *"For the last week our records show that you are engaging really well with the courseware and are well on top of module {{ course }}. Well done you."* This was sent to those in the top 10% of the class while *"For the last week our records show that you are not engaging enough with the courseware for module {{ course }} and you really need to work harder. Please try to make more effort to keep up this week and if you are finding this difficult then do contact the Lecturer."* was sent to those in the lowest 10% of the class, with other 8 custom messages in between.

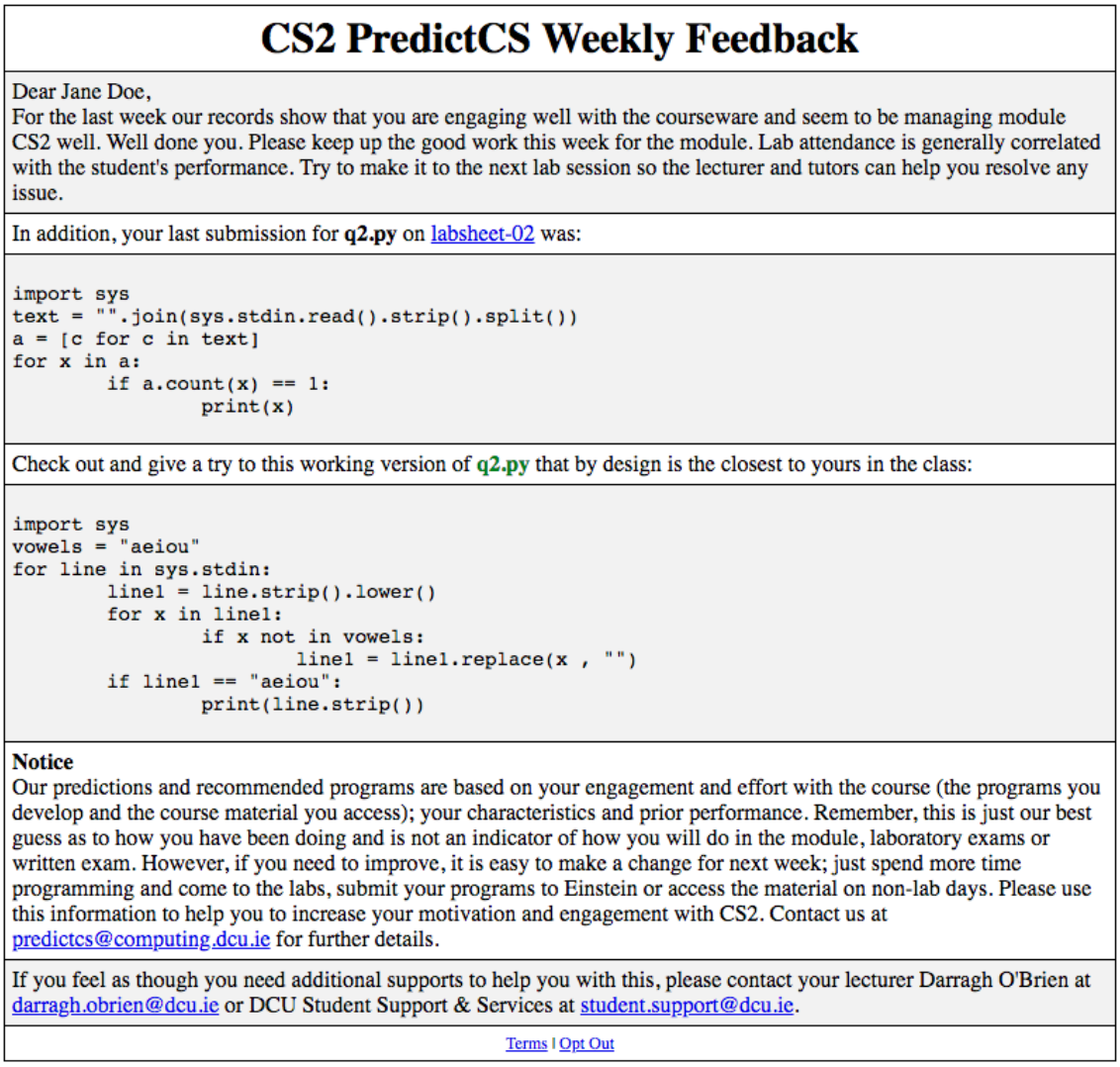
- If a student did not spend any time logged onto the platform and active, we added the message *"Remember, computer programming is a skill that requires practice and this module is no exception."* in bold.

- For each student we checked whether (s)he attended any of the lab sessions the week before and regardless of the predicted probability we acknowledged if they did or we added the message *Try to make it to the next lab session so the lecturer and tutors can help you resolve any issue."* if they did not.

- In addition, as a form of peer feedback, for each notification that was sent we included one computer programming suggestion if the student had submitted a program that failed any of the testcases and was thus incorrect. We developed a knowledge graph and based on the concepts the lecturer considered more important each week for the course, we started suggesting computer programs for those gaps in knowledge. Typically, the most recent labsheet exercises were suggested first, then the previous labsheet exercises and so on. The order of the exercises selected from a particular labsheet is the normal order in the lab. The first exercise in the labsheet will be offered before the second if both were failed submissions for that particular student. The solution suggested is the closest program from a top-rated student in the class that week who got that program working as expected. The top students are the 10% highest-ranked in that class from our predictions each week. We recommended the closest submission by text similarity between the programs after removing comments in the program.

- Students were given an explanation about this project which included sending them alerts, and how the predictions are computed. They were also provided with support resources to reach out to the Lecturer of the module, this project or the Support Services at the University if they needed assistance.

- At the end of the note, students could find links to read the Terms and Conditions for this project, or to unsubscribe from these notifications if desired.



CS2 PredictCS Weekly Feedback

Dear Jane Doe,
For the last week our records show that you are engaging well with the courseware and seem to be managing module CS2 well. Well done you. Please keep up the good work this week for the module. Lab attendance is generally correlated with the student's performance. Try to make it to the next lab session so the lecturer and tutors can help you resolve any issue.

In addition, your last submission for **q2.py** on labsheet-02 was:

```
import sys
text = "".join(sys.stdin.read().strip().split())
a = [c for c in text]
for x in a:
        if a.count(x) == 1:
                print(x)
```

Check out and give a try to this working version of **q2.py** that by design is the closest to yours in the class:

```
import sys
vowels = "aeiou"
for line in sys.stdin:
        line1 = line.strip().lower()
        for x in line1:
                if x not in vowels:
                        line1 = line1.replace(x , "")
        if line1 == "aeiou":
                print(line.strip())
```

Notice
Our predictions and recommended programs are based on your engagement and effort with the course (the programs you develop and the course material you access); your characteristics and prior performance. Remember, this is just our best guess as to how you have been doing and is not an indicator of how you will do in the module, laboratory exams or written exam. However, if you need to improve, it is easy to make a change for next week; just spend more time programming and come to the labs, submit your programs to Einstein or access the material on non-lab days. Please use this information to help you to increase your motivation and engagement with CS2. Contact us at predictcs@computing.dcu.ie for further details.

If you feel as though you need additional supports to help you with this, please contact your lecturer Darragh O'Brien at darragh.obrien@dcu.ie or DCU Student Support & Services at student.support@dcu.ie.

Terms | Opt Out

Figure 6.2: An Anonymised Customised Email Notification Sent to a Student in CA117 during the 2017/2018 Course.

Nobody unsubscribed from these notifications throughout the semester.

See Figure 6.2 for a sample of an anonymised notification sent to a student.

6.3.2 Feedback to Lecturers

Lecturers were sent an associated weekly email indicating the following:

- The percentage of students who attended any of the laboratory sessions that week. This was computed by examining the web logs, the IP associated with those log entries and whether that IP belong to the university or not;

- The percentage of students who had opted-in up to that week to receive customised notifications;
- The average laboratory work completed by students up to that week;
- The amount of time spent on the platform by students on average that week;
- The distribution of the predicted performance for students using the associated probability of our predictions;
- The top-5 most recommended programming submissions that students failed to submit and were recommended by our system;
- Similarly, the top-5 most suggested learning resources;
- Similarly, the top-5 the most suggested laboratory sheets associated to the learning resources.

6.3.3 Measuring Students' Level of Engagement

In this study, we explore how students engaged with the system notifications that include personalised performance messages and resources to focus on based on the students' progression with laboratory work. The recommended resources are suggested by creating a knowledge map with labsheets that are associated to concepts and those concepts are in turn associated with slides to review. For instance, Figure 6.2 recommended that the student work on the first labsheet from week 2 and also to revise the associated material on lists and files.

A difference index for each student i, shown in Equation 6.1, measures the difference between the second examination mark and the first one for a particular student.

$$d_i(e_1, e_2) = e_2 - e_1 \tag{6.1}$$

A gain index was developed to measure each student's improvement between two

examinations, as shown in Equation 6.2, and normalised to output values between -1 and 1 on Equation 6.3:

$$g_i(e_1, e_2) = \frac{(e_2 - e_1)}{e_1} \tag{6.2}$$

$$normg_i(e_1, e_2) = \begin{cases} 0 & e_1 = 0, e_2 = 0 \\ 1 & e_1 = 0, e_2 \neq 0 \\ 1 & g_i(e_1, e_2) > 1 \\ g_i(e_1, e_2) & otherwise \end{cases} \tag{6.3}$$

It is important to note, we should be careful in using our own Equation 6.3 for measuring the impact of our interventions. A student going from an examination grade of 1% to 2% improves 100% with respect to the first examination mark which gives a normalised gain index of 1. However, a student going from 60% to 100% will not have such a high normalised gain index or a student going from 1% to 100% will have its normalised gain index truncated to 1. Results may be skewed due to this approach. An alternative approach to calculate the normalized gain for assessing students' performance in pre- and post-examinations is the g-factor [12].

6.4 RQ4: Quantitative Effects of Adaptive Feedback

We will now analyse the results obtained by running predictions on an incoming cohort of students along with the feedback sent to them and we will examine what this means for the fourth research question proposed in this thesis, RQ4. The type of feedback provided to students has been sent by us across a variety of programming modules including CA116, CA117, CA114, CA277 and CA278 over a period of three academic years.

Earlier in the thesis, in section 1.3 the fourth research question in the thesis was

introduced and it is re-stated here for convenience:

RQ4: What are the effects of timely automatic adaptive support and peer-programming feedback on students' performance in computer programming courses?

We will now examine the impact of this feedback for the three academic years 2015/2016, 2016/2017 and 2017/2018.

6.4.1 Academic year: 2015/2016

In 2015/2016 academic year, students using our system did not receive any notifications. We leverage the data from that year to train our student models. CA278 had not been formed and did not exist at that point as it was taught for the first time in the 2016/2017 academic year. This data from this year is however, used as a baseline to compare to the levels of engagement that happened among students in the following academic years. Even though these would be different actual students form year to year, we believe that because of the size of the classes, the behaviour of students would be acceptably consistent.

Table 6.1 shows some basic characteristics of students who passed and who failed the first assessment in that year. We see that the first assessment had a high failure rate for both courses but there is little differences in terms of age and CAO Points between students that passed or failed this assessment. "Students CAO Route" means how many of the total students for each group that came via their CAO application rather than other routes such as Dare, Access or Sports scholarship.

Table 6.2 shows how students who failed then subsequently improved more than students who passed the first assessment with respect to our normalised gain index. This is usually the case as these students who failed their first assessment have more room for improvement. Note the number of students that passed and failed for each course in Table 6.2 are fewer than the number of students that passed and failed in Table 6.1. That is because for some of the students, their demographics and prior

Table 6.1: Demographics and prior information for students in 2015/2016 in courses CA117 and CA114

Course	Group	Number Students	Mean Age	Students CAO Route	Mean CAO Points
CA117	Passed	66	18.77	49	445.31
	Failed	83	19.07	61	430.08
CA114	Passed	22	18.45	18	387.50
	Failed	49	18.31	42	388.93

information such as route to university could not be retrieved. For instance, that is the case for exchange students. We worked with the university quality office to collect this data but it was not possible for some of the students.

In Table 6.2's *normgi* difference column and in subsequent tables, we calculate a t-test [106] for the means of two samples of scores: either marks of students that passed the first assessment and marks of students that failed, or marks of students that followed an intervention and marks of students that did not, and so on. If we observe a small p-value, we can reject the null hypothesis of equal averages.

6.4.2 Academic year: 2016/2017

In 2016/2017, we trained models using one year of groundtruth data and generated predictions for incoming students that year, that is we trained on data from the year 2015/2016.

In the second part of the semester, we sent customised notifications to students who decided to opt-in to receiving alerts.

We now analyse the effect this type of notification had on students in the 2016/2017 academic year for CA117, CA114 and CA278. For that, we extracted several groups from the students who were enrolled in each course, namely ...

(a) Students who Opted-IN vs. students who Opted-OUT to receive who customised notifications;

Table 6.2: Difference and Normalised Gain Index between the examinations for CA117 and CA114 in the 2015/2016 academic year

Course	First Exam Week	Second Exam Week	Group (Number)	First Exam	Second Exam	*di* Mean (Std.Dev.)	*normgi* Mean (Std.Dev.)	*normgi* Difference
CA117	W6	W12	Passed (66)	75.23 (20.08)	55.06 (29.94)	0.00 (36.51)	-0.28 (0.36)	-0.54**
			Failed (83)	14.70 (13.65)	24.40 (24.74)	9.70 (23.37)	0.26 (0.71)	
CA114	W6	W12	Passed (24)	64.17 (22.35)	64.17 (34.15)	36.08 (34.07)	0.05 (0.59)	-0.6**
			Failed (51)	5.88 (9.11)	41.96 (31.75)	22.81 (37.71)	0.65 (0.65)	

** $p - value < 0.001$

(b) Students who Fixed vs students who Did-not-fix the programs they were suggested in the notifications.

(c) Students who Passed vs students who Failed their first laboratory exam.

On the last division, students who passed or failed the first laboratory exam, contains all the students in the class. However, the other two divisions do not. Students had to opt-in or opt-out before they could submit any more programs to be analysed in the grading platform, but some of them were not engaged and did not reply. In addition, there are some students who were not sent notifications, were deemed to have failed in their programming submissions as they did not have any to submit, so they were not in the fixed nor in the did-not-fix group.

Table 6.3 shows some characteristics of those students in the different groups for the courses being analysed. There are no major differences in between the groups. In general, students who passed the first assessment for all courses have higher CAO points if they came to University through that route. Also, in general, most students would opt-in and would not fix the programs suggested in their notifications.

Table 6.4 shows the differences among the groups with respect to examinations. Examinations happened in Week 6 (mid-semester) and Week 12 (end-of-semester classes). Again, students who failed the first assessment had more room for improvement than students who passed that assessment, for all courses. In general, for this course and this academic year, there are only significant differences for those students who opted in for CA117's notifications and not for the other courses with respect to the opt-outs. However, students who fixed the programming submissions that were suggested in the notifications, improved more using the normalised gain index with respect to students who did not fix their programs for all courses: CA117, CA114 and CA278. Again, we do not have demographics and prior information for all the students. This can be seen as there are fewer students in the pass and fail groups for Table 6.3 with respect to Table 6.4.

Table 6.3: Demographic information and prior information from the 2016/2017 student groups in CA117, CA114 and CA278

Course	Group	Number Students	Mean Age	Students CAO Route	Mean CAO Points
CA117	Passed	81	18.88	62	438.79
	Failed	56	18.75	41	399.15
	Opted-IN	119	18.82	91	427.25
	Opted-OUT	11	18.82	9	446.67
	Fixed	16	18.62	12	438.75
	Did-not-fix	51	18.53	38	398.16
CA114	Passed	56	18.34	47	410.32
	Failed	16	18.25	14	396.79
	Opted-IN	62	18.31	51	401.37
	Opted-OUT	7	18.14	7	457.14
	Fixed	18	18.44	13	415.00
	Did-not-fix	34	18.15	29	405.86
CA278	Passed	52	18.13	42	411.79
	Failed	5	18.00	5	404.00
	Opted-IN	41	18.24	36	414.17
	Opted-OUT	9	17.78	6	393.33
	Fixed	7	18.29	4	450.00
	Did-not-fix	27	18.30	25	403.20

Table 6.4: Difference and Normalised Gain Index among the examinations for CA117, CA114 and CA278 in the 2016/2017 academic year

Course	Group (Number)	First Exam	Second Exam	*di* Mean (Std.Dev.)	*normgi* Mean (Std.Dev.)	*normgi* Difference
CA117	Passed (82)	76.22 (21.70)	47.85 (26.42)	-28.37 (22.28)	-0.38 (0.30)	-0.56**
	Failed (58)	8.62 (11.88)	12.02 (14.64)	3.40 (12.79)	0.18 (0.53)	
	Opted-IN (122)	54.51 (36.36)	37.75 (27.42)	-16.76 (25.51)	-0.15 (0.52)	+0.18
	Opted-OUT (11)	52.27 (34.47)	19.45 (17.07)	-32.82 (28.52)	-0.33 (0.68)	
	Fixed (16)	32.81 (39.25)	27.62 (25.45)	-5.19 (23.28)	0.23 (0.68)	+0.37*
	Did-Not-Fix (52)	45.19 (31.79)	34.42 (27.46)	-10.77 (21.89)	-0.14 (0.45)	
CA114	Passed (57)	72.81 (18.90)	55.44 (29.02)	-17.37 (31.21)	-0.20 (0.46)	-0.55**
	Failed (16)	17.19 (11.59)	40.00 (29.15)	22.81 (37.71)	0.35 (0.69)	
	Opted-IN (62)	62.10 (28.31)	53.23 (29.17)	-8.87 (37.09)	-0.11 (0.54)	-0.04
	Opted-OUT (7)	57.14 (25.75)	42.86 (32.83)	-14.29 (38.68)	-0.07 (0.74)	
	Fixed (18)	62.50 (29.17)	56.67 (28.48)	-5.83 (36.83)	-0.03 (0.51)	+0.12
	Did-Not-Fix (34)	61.03 (22.84)	50.59 (29.99)	-10.44 (32.21)	-0.15 (0.54)	
CA278	Passed (53)	62.57 (17.65)	73.91 (17.44)	11.34 (19.87)	0.23 (0.36)	-0.73**
	Failed (5)	15.80 (12.94)	64.80 (8.13)	49.00 (17.45)	0.96 (0.09)	
	Opted-IN (42)	60.88 (23.24)	76.45 (14.91)	15.57 (23.74)	0.31 (0.41)	-0.01
	Opted-OUT (8)	58.12 (19.81)	71.38 (11.79)	13.25 (13.45)	0.32 (0.32)	
	Fixed (7)	53.57 (27.49)	73.00 (14.90)	19.43 (33.00)	0.41 (0.50)	+0.11
	Did-Not-Fix (28)	58.25 (20.88)	73.93 (14.90)	15.68 (22.12)	0.30 (0.39)	

* $p - value = 0.01$
** $p - value < 0.001$

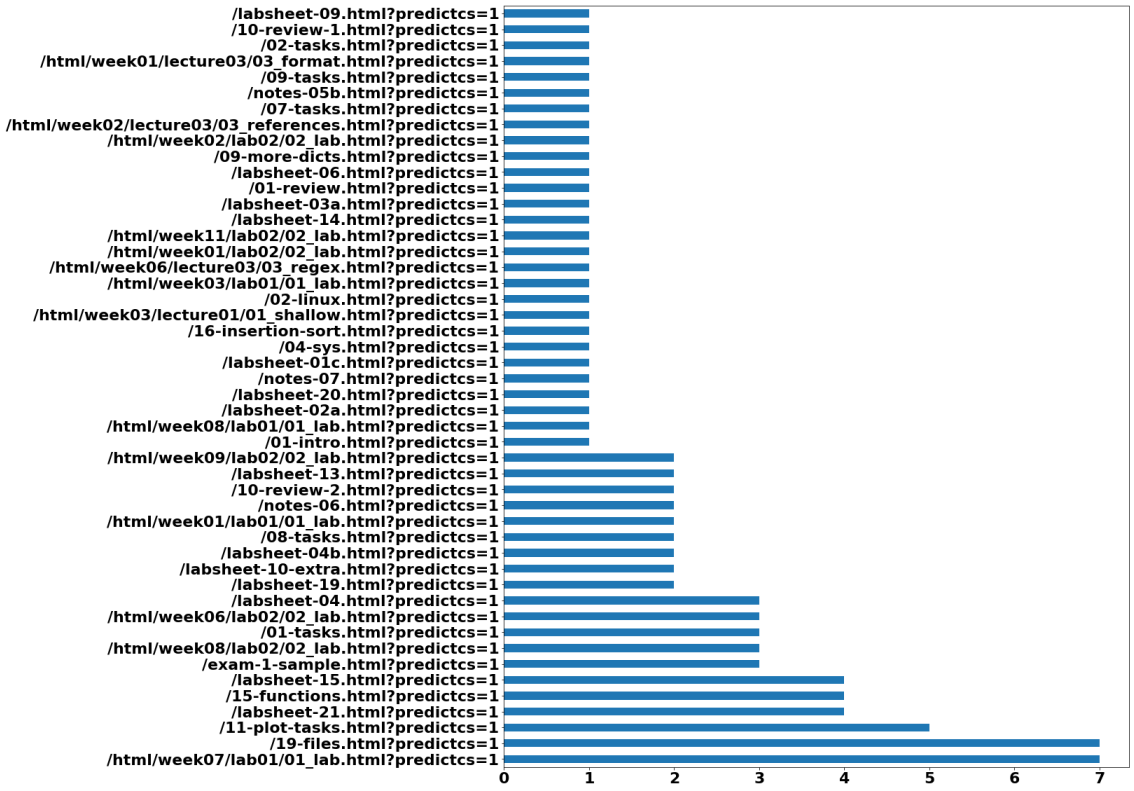


Figure 6.3: Frequency of Access to Material and Labsheets from the Notifications

6.4.3 Academic year: 2017/2018

In 2017/2018, we trained models using two years of groundtruth data and we generated predictions for incoming students in that year. Again, in the second part of the semester, we sent customised notifications to students who decided to opt-in. In addition, in this 2017/2018 academic year, we measured whether a student would click the material resources suggested on the notifications. Figure 6.3 shows how we tracked when students accessed the material or labsheets from the links in the notifications.

Table 6.5 shows that students who clicked on any resource (which were not a majority) showed a greater normalised gain than students who did not click on those resources between examinations. Examinations happened in Week 6 and Week 12 as in previous years. The normalised gain difference between the two groups is found to be statistically significant.

Table 6.5: Difference and Normalised Gain Index between the examinations for CA117, CA114 and CA278 on 2017/2018 academic year

Course	Group (Number)	First Exam	Second Exam	*di* Mean (Std.Dev.)	*normgi* Mean (Std.Dev.)	*normgi* Difference
CA117	Passed (90)	75.83 (20.90)	55.11 (28.53)	-20.72 (23.05)	-0.28 (0.33)	-0.59**
	Failed (58)	12.93 (12.49)	25.00 (26.08)	12.07 (24.74)	0.31 (0.67)	
	Clicked (19)	51.32 (39.30)	53.16 (24.72)	1.84 (28.20)	0.22 (0.58)	+0.31[1]
	Did-not-click (129)	51.16 (35.06)	41.86 (31.86)	-9.30 (28.41)	-0.09 (0.56)	
CA114	Passed (53)	83.02 (21.60)	79.62 (27.88)	-3.40 (32.96)	0.04 (0.47)	-0.38*
	Failed (16)	10.94 (12.40)	40.00 (36.74)	29.06 (34.52)	0.42 (0.71)	
	Clicked (4)	50.00 (30.62)	90.00 (10.00)	40.00 (35.88)	0.70 (0.52)	+0.61[1]
	Did-not-click (65)	67.31 (36.41)	69.23 (35.10)	1.92 (34.86)	0.09 (0.55)	
CA278	Passed (70)	69.06 (16.34)	65.99 (17.24)	-0.01 (0.27)	65.99 (17.24)	-0.01**
	Failed (10)	31.90 (10.71)	50.00 (22.03)	0.47 (0.37)	50.00 (22.03)	
	Clicked (5)	66.40 (21.14)	60.00 (17.99)	-0.08 (0.14)	60.00 (17.99)	+0.11
	Did-not-click (75)	64.28 (19.89)	64.25 (18.68)	0.05 (0.33)	64.25 (18.68)	

[1] $p - value = 0.03$
* $p - value = 0.01$
** $p - value < 0.001$

6.4.4 Comparison with the baseline

Table 6.6 shows a comparison between students that passed and failed the first laboratory exam in 2015/2016, 2016/2017 and 2017/2018 academic years. Recall, in 2015/2016, there were no interventions to students; in 2016/2017 and 2017/2018 there were customized notifications sent to students. The normalized gain index difference between students that passed and failed the first examination is reduced for CA114 and slightly increased in CA117. We can not state students that are lower performers in the first examinations get their difference reduced with respect to the higher performers unless they engage with these notifications by fixing their suggested failed submissions or clicking on the resources suggested.

Table 6.6: Comparision between 2015/2016, 2016/2017 and 2018/2019 academic years

Course	Academic Year	Group (Number)	*normgi* Difference
CA117	2015/2016	Passed (66)	-0.54**
		Failed (83)	
	2016/2017	Passed (82)	-0.56**
		Failed (58)	
	2017/2018	Passed (90)	-0.59**
		Failed (58)	
CA114	2015/2016	Passed (24)	-0.6**
		Failed (51)	
	2016/2017	Passed (57)	-0.55**
		Failed (16)	
	2017/2018	Passed (53)	-0.38*
		Failed (16)	

* $p - value = 0.01$
** $p - value < 0.001$

In terms of addressing **RQ4**, re-stated earlier at the start of this section, the effects of engaging with these personalised notifications by fixing programming sub-

missions or clicking on learning material might have a positive effect on the progression from one examination to the next one[1]. This means the research question is addressed with a positive answer.

We believe it is worth the effort of making this type of interventions and putting our predictive modelling work to practice. We feel morally compelled to implement this type of interventions and we can potentially guide, help and motivate our student body. In the future, we could explore how students engage with the programming code solutions from higher performers and how it affects their programming design learning.

6.5 RQ5: Qualitative Feedback

In section 1.3 earlier in this thesis, the fifth research question in the thesis was introduced and it is re-stated here for convenience:

RQ5: What are students' and teachers' perspectives and experiences after adopting a predictive modelling and adaptive feedback system into their own classes ?

6.5.1 Students have their say

In any student intervention, it is important to capture students' opinions regarding the feedback shared with them, to understand how it affects their behaviour within the modules and whether it encourages them to try new solutions to their programming assignments or to revise course material. Thus, we gathered students' opinions about our system and the notifications sent to them via a written questionnaire at the end of the semester. The questions on the form were listed as follows:

- **Q1:** Did you opt-in ? [Yes / No]
- **Q2:** If you opted-out, could you tell us why ? [Comment]

[1]The code for this work has been made available as a GitHub repository at https://github.com/dazcona/edm-engagement

- **Q3:** How useful did you find the weekly notifications ? [1 to 5 star rating]
- **Q4:** Did you run any of the working programs suggested to you ? [Yes / No / I was never suggested any]
- **Q5:** Would you recommend the system to a student taking this same module next year ? [Yes / No]
- **Q6:** Would you like to see weekly the system notifications for other modules ? [Yes / No]
- **Q7:** How could we improve the system for next year ? Any other comments. [Comment]

Overall, feedback from students in these student surveys was very positive from students and a summary of the responses can be found in Table 6.7 for the 2016/2017 academic year. The survey results were anonymised. Most students would recommend this system to other students attending the same module next year or would like to see this system included in other modules as shown in answers to questions 5 and 6 respectively.

Table 6.7: 2016/2017 Student survey responses from students about the project

Course	Response Rate	Q1	Q3	Q4	Q5	Q6
CA114	80.25%	84.60%	3.82 ⋆	40.40%	91.33%	91.38%
CA117	75.71%	93.40%	3.49 ⋆	33.70%	85.15%	83%
CA278	53.33%	87.50%	3.45 ⋆	21.88%	100%	90.00%

For first-year courses, the questionnaire was completed on the second-last day of the semester classes during an evaluation for another module where all CA and EC first-year students should have attended. We could gather a good number of responses. The response to the first question shows the percentage of students who opted-in and the fourth shows whether they ran any of the suggested programs. The responses in both questions are inaccurate as, for instance, some students claimed

they opted-in when they did not or they were not suggested any program when in reality they were. That indicates that some students may not check their mail regularly or they opted-in without realising what they were signing up for or what the system would do with their digital footprint. Notifications via email may not be the best way to communicate with students as some of them pointed out in the improvements and comments section of the survey and a better way to measure how they interact with these customised messages should be tried.

The final question regarding how students would like to improve the system received some really interesting comments. However, students who were doing well or very well, were getting a similar response every week and the notifications might seem monotonous. In general, students demanded a more personalised notification as well as some additional learning resources. Finally, the following are some positive and negative quotes, comments and suggested improvements from students in response to this last question of the survey:
A significant number of students demanded more personalised feedback:

- "More detailed responses, where you can improve"
- "More varied responses"
- "Give more personalised feedback"
- "Maybe more detailed feedback"
- "More in depth analysis of progress, more precise areas to focus on"

Some students also asked for other features:

- "Feedback on each step of each task would be good, such as, better ways to do things"
- "Have it from the start of the year"
- "Send a(n) end of module feedback of the whole module"

- "Maybe more suggested working programs"
- "Give advice in what the student is performing poor in"

Others did not enjoy the notifications that much as they could be very repetitive for students who are well on top of the module and do not have any failed programs for which they could receive suggestions:

- "Less repetition / automation. (It) would be nice to receive other feedback besides "you are doing OK"."
- "Always said the same thing"
- "More information relevant to how you are doing, it is too vague"

Overall the feedback was very positive and some students were motivated with the weekly notifications:

- "It gave me confidence about the module. It gave me reassurance as to how I was getting on."
- "Good service, very helpful and effective way to manage your module"

6.5.2 Lecturers have their say

In addition to providing feedback directly to students, we enabled a discussion among the lecturers of the modules that adopted this system and formal feedback was sent to researchers via email.

CA116 and CA114's lecturers also completed a survey and indicated the following:

1. "With large class sizes for our programming modules, it is practically impossible for the Lecturer to monitor each student personally. Therefore, an automated approach is useful".

2. "Simply seeing the list of students marked red or green each week gives a sense for how things are going".

3. "Perhaps I'm mistaken, but the predictions seem too negative".

4. "I calibrate the difficulty of mid-semester lab exams depending on what has been taught, when and how the students are doing. PredictCS doesn't do a great job of talking this "human factor" into account".

5. "Most students respond positively to some of the gamification aspects of the system, e.g. progress graphs. PredictCS doesn't (yet) take advantage of that aspect of how students engage".

CA117's lecturer said "(He) would be happy for you to run further experiments in future deliveries of CA117". "(He) liked the fact you could tailor the release of concepts to students in order to keep pace with the delivery of the module e.g. avoiding sharing solutions by advanced students that used lambda expressions not yet covered". The following year, he indicated that "(He) likes the fact it does its thing without my input. He does use it for checking overall interaction by students with lab exercises and that information he finds valuable.".

CA278's lecturer said "I found the PredictCS to be a nice complement to the lectures and worksheets. The feature which suggests similar programs to a student's attempt is good although it would probably work better in situations where there isn't a weekly deadline for the exercises and where the solutions aren't provided after the deadline (as was the case with CA278)".

As a summary for addressing **RQ5**, the platform was well received by students and lecturers. Students would like to see their feedback in other modules and would recommend it to other students taking those courses in the following year. Lecturers show the recommendations as an extra help to automate the personalisation of the experience of hundreds of students.

6.6 Extra: Virtual Coding Assistant

Conventionally, Learning Analytics are used to process student data for a variety of possible applications including feedback to University administrators, or to notify students regarding their predicted performance and available further resources using email or via a university's Learning Management System.

In the 2018/2019 academic year, to support students to engage in learning and to become more pro-active and responsible about their own learning, we designed and tested a system called CoderBot. CoderBot is an Artificial Intelligence Chatbot service deployed on WhatsApp[2] as a coding assistant to support learning of computer programming. CoderBot[3] was deployed in CA116. Students were able to interact with the assistant and find out the following:

- Personalised messages about their predicted performance in examinations and assessments: A Predictive Machine Learning classification model is built by aggregating multiple sources of student data (academic, programming work, and logged interactions with offline and online resources), handcrafting features and extracting patterns of success on the course leveraging Artificial Intelligence techniques. The model is trained with two years of ground-truth data and cross-validated on the training data. Predictions are generated weekly for the new cohort of incoming student data. Using the classification probabilities, we divide students into deciles and designed a message for each group.

- Recommended material: Students are suggested material to help their learning such as slides and exercises they might want to check out based on their progression and effort on the course.

- Short code snippets: Students can avail of code snippets that showcase functionality such as slicing lists, reading from files or printing arguments. 100+

[2]WhatsApp Messenger is a freeware and cross-platform messaging and Voice over IP service owned by Facebook

[3]Code assistant has a separate repository at https://github.com/dazcona/code-assistant

snippets have been hosted on GitHub's gists, as that website is already optimised for easy reading of programming code on smartphones.

In addition to the above, students can ask for further help from the Lecturer or the University's support services, consult the terms of the project and opt-out at any time. Phone numbers used for WhatsApp are deleted at the end of the semester. Efforts are now being made to include Natural Language Understanding so students can ask questions in natural language such as "How am I doing?" or "What can I learn next?"

In 2018/2019, in terms of **RQ4**, after running the same quantitative analysis between students that engaged and talked to the virtual assistant (52 students) showed a greater normalised gain than students that did not talk to the chatbot (80 students). The normalised gain difference between the two groups is not found to be statistically significant. Engaging witht these type of feedback was found to have an effect and many more students engage with the chatbot than with previous email notifications.

Chapter 7

Using Graph Theory and Networks to Model Students

7.1 Introduction

In this chapter, in order to address the final of the 6 research questions which are the basis of this thesis, we collected a unique dataset of college student learning states and navigation traces from student access logs to a large MOOC. We present a straightforward way for researchers and lecturers to quickly follow up on a particular student's progression using networks. Students have different strategies to study concepts and that can be exploited to personalise each student's learning. The potential application of unsupervised machine learning approaches such as HMMs using deep learning might showcase hidden and higher level representations of learning states that can be applied to Intelligent Tutoring Systems and online courses, and we explore that further in this Chapter.

7.2 The Global Freshmen Academy at Arizona State University

Arizona State University (ASU)'s Global Freshman Academy (GFA)[1] provides first-year university courses through the online MOOC platform, EdX. EdX is a massive open online course (MOOC) provider which supports access to online courses from numerous Universities worldwide. It was formed by a coalition of MIT and Harvard University but has opened up to host courses from many other Universities, in a range of topics. The GFA was launched in 2015 in partnership with EdX and since its launch it has enrolled more than 230,000 students from more than 180 countries.

The GFA offers a wide range of courses in Business, Engineering and General studies and in our analysis we will focus on only a small subset of these, specifically the following two Mathematics modules:

- **College Algebra and Problem Solving I, MAT117**: this online algebra course equips students with the skills to effectively solve problems using algebraic reasoning. Students learn about systems of linear equations, rational functions, quadratic functions, logarithmic functions, general polynomial functions, and exponential functions.

- **Precalculus, MAT170**: this online calculus course focuses on quantitative reasoning and functions. Students develop the skills to describe the behaviour and properties of linear, exponential, logarithmic, polynomial, rational, and trigonometric functions. Before taking this course, students should already have a strong understanding of algebraic skills such as factoring, basic equation solving, and the rules of exponents and radicals, which can be mastered through the college algebra course.

Additionally, both the Algebra and Precalculus courses uses the cutting-edge adaptive technology ALEKS. ALEKS is a personalized math tutor that helps students

[1] https://gfa.asu.edu/

learn mathematics skills at their own pace. The courses tailor content and personalizes the learning experience around the student's skill level, allowing the student to achieve mastery in a certain concept before moving on to the next. Utilizing the ALEKS learning system, students expect to be instructed on the topics they are most ready to learn. Further details regarding the ALEKS platform for this study can be found in Chapter 3. The log data we extracted when students are assessed continuously while navigating through ALEKS is summarized in Table 7.1.

Table 7.1: Log Data extracted from EdX & ALEKS

Number of Occurrences	Description
16,022	students
40,356	assessments
8,808,675	daily aggregate events of the topics learned and retained
186,224	students mastering topics
5,022,091	transactions of students navigating through the concepts

As a further breakdown into this dataset, the following is the information which was provided to us for this analysis and this stored by us in a non-relational (NoSQL) database for easy access to explore and analyse:

- Assessments: records of students being assessed, the number of topics mastered and the reason of the examinations; the reasons for being assessed are the following:
  - Initial Knowledge Check;
  - Progress Knowledge Check;
  - Objective Completion;
  - Login Time Knowledge Check;
  - Periodic Knowledge Check;
  - Class Completion Knowledge Check;

– Skipped Knowledge Check.

For instance, a student is assessed when she first comes onto the platform as an Initial Knowledge Check, as a pretest given at the beginning of the course, and the content shown afterwards depends on the topics the student already knows. Also, students are assessed periodically with a Progress Knowledge Check to see how many new topics have been mastered by them. See the numbers for each reason of assessment in Figure 7.1;

- Daily rollup: a daily progress record regarding the topics mastered by each student and the percentage goal towards completion of the course. For instance, for a student that has some activity on a particular day a list of topics retained and learned are calculated and stored as a daily row in this collection;

- Concepts: topics students work on, each of which belong to sections. For instance, topic "Evaluating functions: Linear and quadratic or cubic" which belong to Section 4 of the course MAT117x;

- Transactions: fine-grained records of students navigating through the various concepts and sections of the courses showing the activity types and duration for each. For instance, a student is reading the Lesson for 25 seconds for the topic "Evaluating a quadratic expression: Integers" which belongs to the first section of the course MAT117x. We will dig deeper into this transactional data.

We carried out a range of sanity checks to ensure the data we are working with did not have any gaps. The data provided was from 13th. April 2016 to 1st. October 2017. However, we discovered there were no transactions or assessments data in the Summer of 2017. We believe there could have been a data breach at that time but it is unclear and the end result is that we can not make any inferences of what happened during that time or even after when the data collection was resumed. In order to progress with our analysis, we removed the data we have after the data

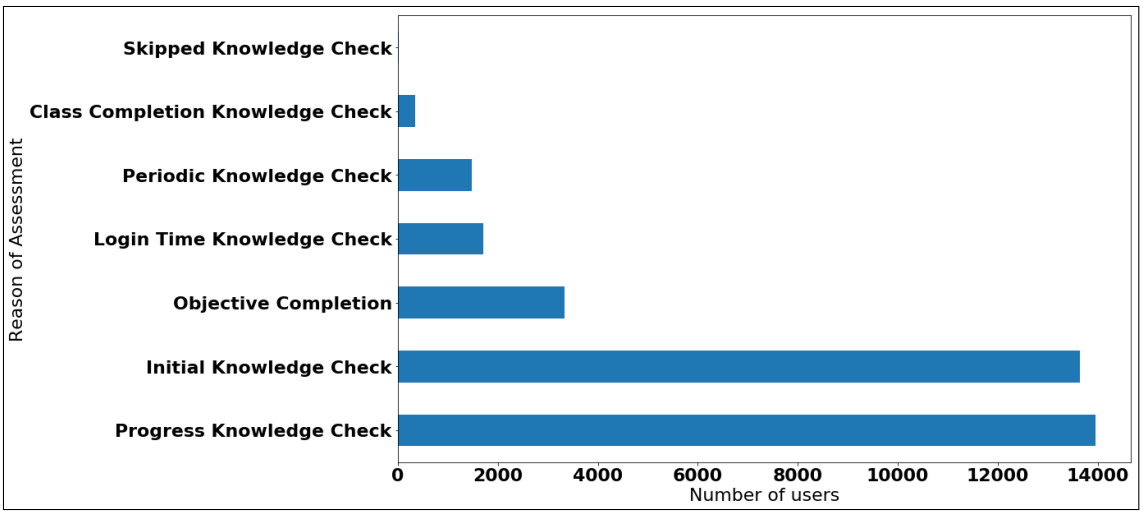


Figure 7.1: Number of Students that Took each Type of Assessment for MAT117 and MAT170 in ASU's GFA via EdX

breach for the retrospective analysis. This the actual dataset that we used was until August 2017.

7.3 Exploratory Data Analysis

We explored the data sources provided such as assessments, transactions or mastery of concepts. Figure 7.2 shows the number of students and the percentage of completion for each course. This graph has been developed by looking at the mastery of concepts data source collection. Most students in our dataset were enrolled in MAT117. As in other MOOCs, many students dropped out in the beginning after completing only a small percentage of course content. Fewer and fewer students complete the subsequent parts of the course. For MA117, around 400 students completed the course entirely.

In a student's learning journey on the MOOC the system navigates them to different concepts or topics based on their understanding and performance in previous topics. For instance, some of the topic names are "Distributive property: Integer coefficients" or "Writing and evaluating a function modeling continuous exponential growth or decay given two outputs". In any topic, students generally start with the initial learning state of reading the lesson (marked with an L), then their under-

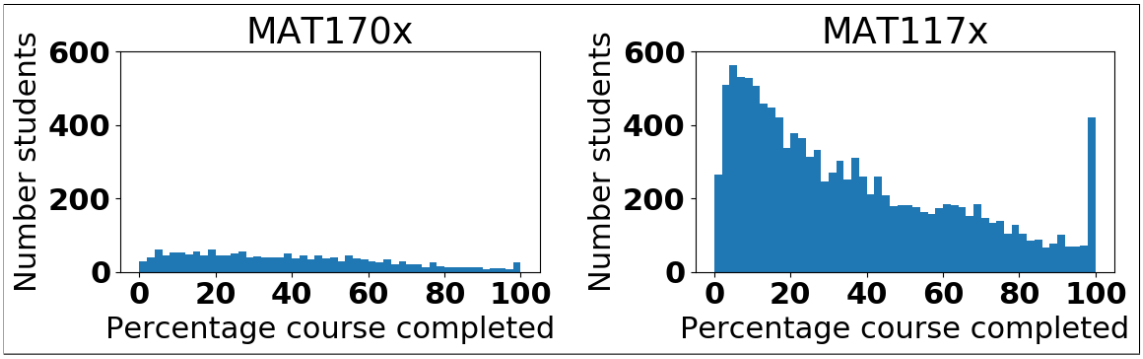


Figure 7.2: Number of Students Binned Based on their Completion Percentage of Each of the Courses

standing and mastery is evaluated via some exercises that they can get correct (C) or wrong (W). They could also request a working example of the concept (E). After some work, the system marks the understanding of the students as a provisional mastery (S) or a failure to learn a concept (F). The student is then redirected to another concept.

There are a total of 797 topics in the 2 courses and Figure 7.3 shows the time spent by all students in each learning state, shown in seconds using the transactional data, ordered by the total time spent by all students on that topic.

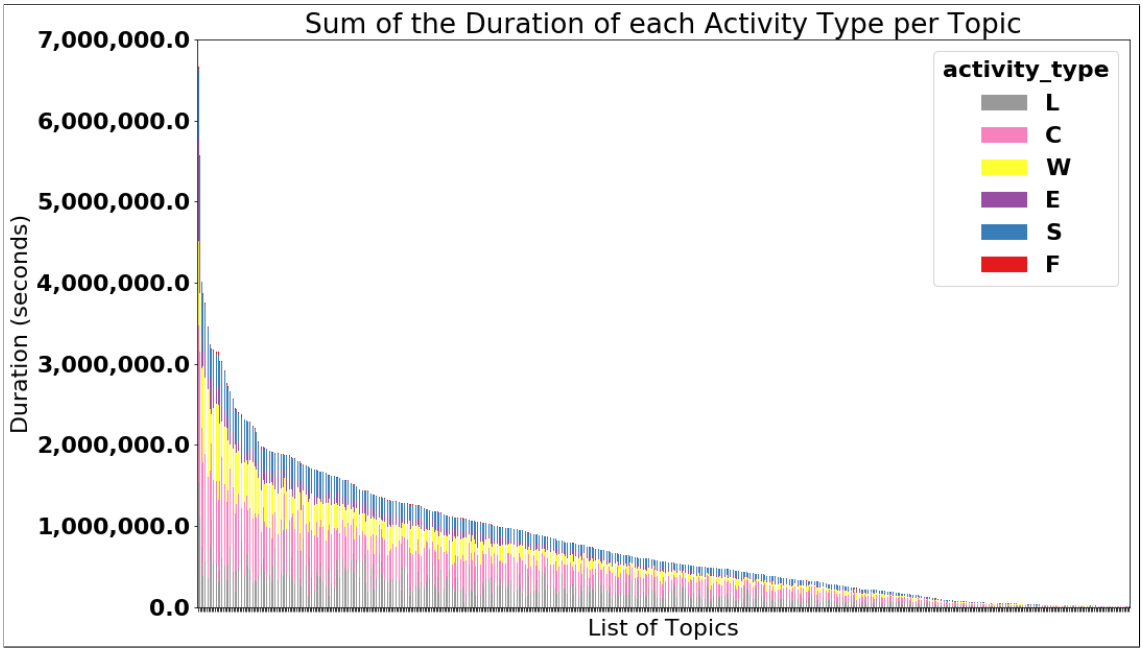


Figure 7.3: Total Duration for each Topic Using All Students' Data, Ordered, Split and Colour Coded for each Learning State

In both courses, we encounter 8 different sections. Each topic corresponds to one of these sections (also known as slices). Sections are higher level representations of topics. Figure 7.4 shows the numbers of students who progressed and worked through each section. This graph shows only 40.07% reached to the second section. Based on the student's understanding in each of the concepts belong to a particular section and the assessments taken, students are able to progress between sections. Only 0.44% of the students that started reached to the final section, section 8.

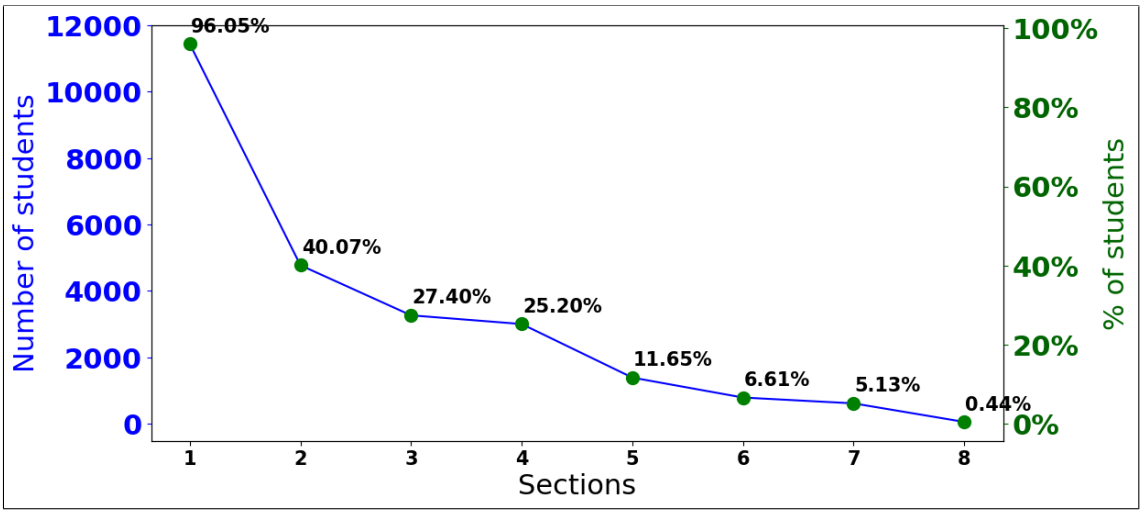


Figure 7.4: Number and Percentage of Students Who Worked on Each Section for Both Courses

We developed a web application to explore this dataset and, specially, the transactions recorded while students move from one learning state to another, one topic to another and one slice or section (higher-level representations of concepts) to another. The EdX and ALEKS log data was stored and then indexed in a non-relational database. This information was made available using our web application where Faculty can explore individual students and how they are redirected through the material, concepts and their underlying difficulty based on completion and likely sequence patterns around concepts and slices.

The application uses D3.js and NetworkX to visualise transactions interactively for each student and in between concepts and slices[2]. For instance, the first trans-

[2]The code used for this project has been made available as a GitHub repository in https://github.com/dazcona/edm-networks

actions for a particular student are shown in Figure 7.5. We can explore all this student's transactions grouped by the dates. Figure 7.6 which shows the time distribution for each learning state for this student with respect to the other states. In addition, we can create an interactive visualization for each of the topic's transactions using networks. See Figure 7.7. We can appreciate differences between simpler concepts such as "Ordering integers" (shown in Figure 7.6) and more challenging ones such as "Evaluating a linear expression: Integer multiplication with addition or subtraction" (shown in Figure 7.7). This will be further analysed in the next section utilizing graph theory.

ID	Timestamp	Topic	Duration	Activity	Section
5ac17b2177b2f1156d5caf88	2017-06-11 11:32:44	Ordering integers	21	L	1
5ac17b2177b2f1156d5caf89	2017-06-11 11:33:16	Ordering integers	0	E	1
5ac17b2177b2f1156d5caf8a	2017-06-21 17:10:54	Ordering integers	206	E	1
5ac17b2177b2f1156d5caf8b	2017-06-21 17:15:09	Ordering integers	49	W	1
5ac17b2177b2f1156d5caf8c	2017-06-21 17:16:03	Ordering integers	54	C	1
5ac17b2177b2f1156d5caf8d	2017-06-21 17:16:28	Ordering integers	25	C	1
5ac17b2177b2f1156d5caf8e	2017-06-21 17:16:42	Ordering integers	14	S	1
5ac17b2177b2f1156d5caf3a	2017-06-21 17:34:15	Evaluating a linear expression: Integer multiplication with addition or subtraction	1022	L	1
5ac17b2177b2f1156d5caf3b	2017-06-21 18:14:52	Evaluating a linear expression: Integer multiplication with addition or subtraction	0	E	1
5ac17b2177b2f1156d5caf3c	2017-06-21 18:15:34	Evaluating a linear expression: Integer multiplication with addition or subtraction	6	E	1
5ac17b2177b2f1156d5caf3d	2017-06-21 18:18:51	Evaluating a linear expression: Integer multiplication with addition or subtraction	191	W	1
5ac17b2177b2f1156d5caf3e	2017-06-21 18:20:18	Evaluating a linear expression: Integer multiplication with addition or subtraction	87	W	1
5ac17b2177b2f1156d5caf3f	2017-06-21 18:20:18	Evaluating a linear expression: Integer multiplication with addition or subtraction	49	E	1

Figure 7.5: Screengrab from the Web Application which shows the First Transactions for a Particular Student

7.4 RQ6: Insights from MOOCs and Sequences of Learning States

In section 1.3 the sixth research question in the thesis was introduced and it is restated here for convenience:

RQ6: Can we extract valuable insights from massive open online learning platforms utilising the sequences of learning states?

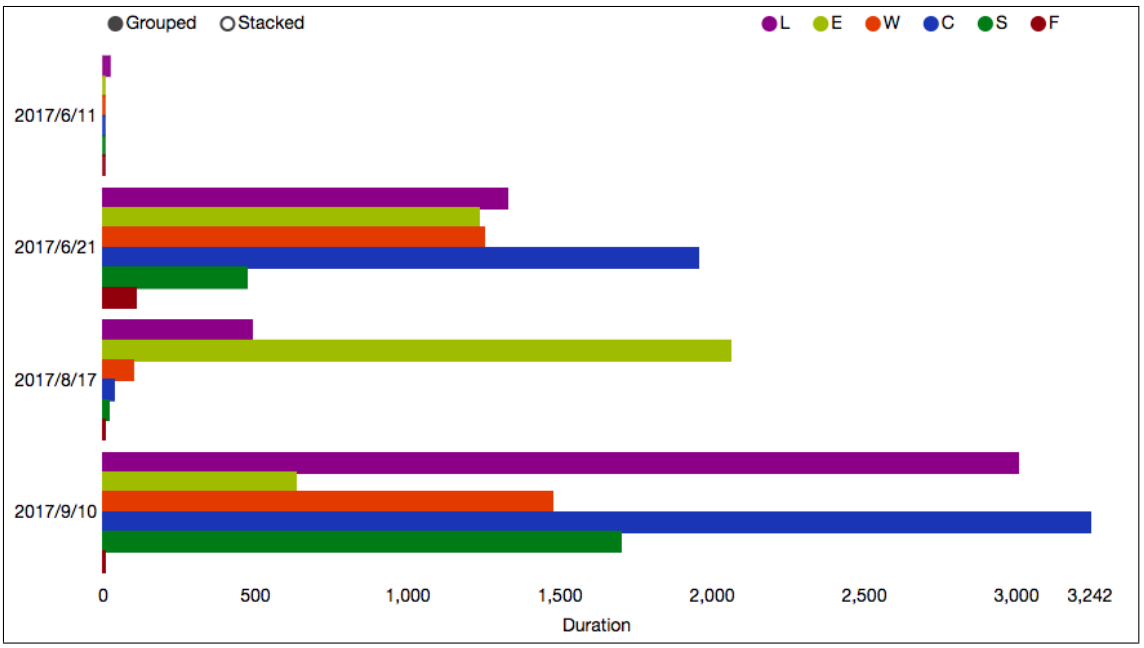
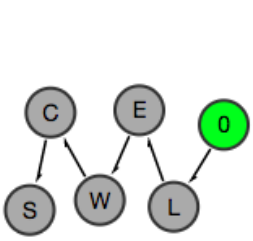
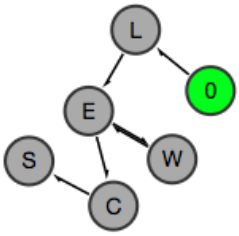


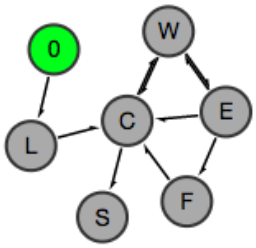
Figure 7.6: Screengrab from the Web Application which shows the Distribution of Learning States Grouped By Date for a Particular Student



(a) Network Visualisation for Topic "Ordering integers"



(b) Network Visualisation for Topic "Evaluating a linear expression: Integer multiplication with addition or subtraction"



(c) Network Visualisation for Topic "Exponents and integers: Problem type 1"

Figure 7.7: Screengrab from the Web Application which shows Network Visualisations for the First Three Topics for a Particular Student

Of the 6 research questions posed in this thesis, this one is probably the most open-ended in that the question does not have an obvious yes/no answer and requires us to experiment with, and find, the so-called "valuable insights". This requires a different approach to what we have done in earlier chapters, namely building predictive models of student outcomes and deploying them in practice, or using them as a form of factor analysis to explore features which influence student performance. What

we did here is to try to learn useful structure but without labelled classes as we had in the previous chapters (student exam performance being the labels). This is also known as unsupervised learning and we proceeded in this work by deriving graph representations of the transactional data logs from the MOOC. The 5 million transactions contain the following information: student, timestamp, concept, learning state and duration of the activity and this formed the basis for the investigation.

Through the ALEKS platform, we are able to observe the learning states students will go through as they consume content through the GFA platform. We analysed how students transition from one learning state to another and as an example, Figure 7.8 shows how a particular student traversed through the learning states for two particular topics: "Ordering integers" and "Exponents and integers: Problem type 1". Even for the same student, there are different learning paths being taken for different topics and we can see this by examining the learning states. Figure 7.8 shows two very different learning paths, a linear learning path versus the student failing to master the concept at the beginning but being brought back to that concept later on and mastering it eventually.

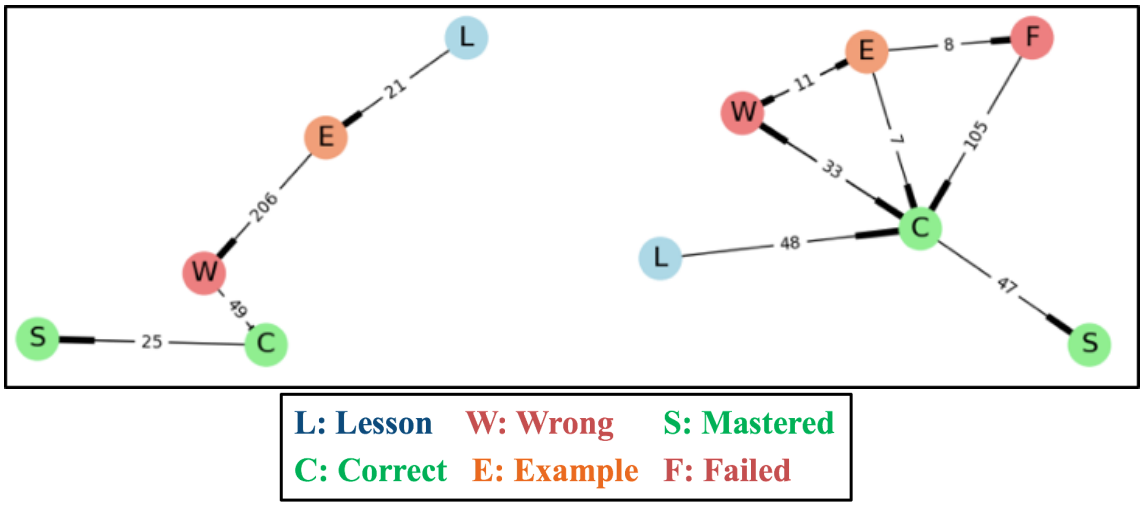


Figure 7.8: Visualizations of Networks for Two Students Going Through Various Learning States on Two Different Topics

Topic Networks

Students are navigated through topics, also known as concepts, by using the ALEKS technology. We developed directed networks for each topic by using the data from all students. There are 413 topics in MAT170 and 384 in MAT117. For instance, a student can follow the learning path "LWCEWCECS" for a particular topic such as "Ordering integers".

For each topic, a graph was developed using the following parameters:

1. Between pairs of learning states (pair-wise and including states to themselves):
    - Count: number of edges between two learning states;
    - Sum: adding up the duration of time spent between two learning states;
    - Average: the average duration of time spent between two learning states e.g. L-C-avg_duration;
    - Std: standard deviation of the durations between two learning states.
2. For each learning state we extracted the following in-coming and out-going metrics:
    - Total Count of in-coming or out-going edges;
    - Total Sum of the duration of the in-coming or out-going edges;
    - Weighted Average of in-coming or out-going durations e.g. in-coming-L-avg;
    - Weighted Standard Deviation of in-coming or out-going durations.

Once this data had been extracted, metrics were derived for each learning state in a topic network based on the in-coming and out-going metrics. These metrics derived for a topic network can be compared with other topic networks using the network or graph topology alone. We select two topic networks to illustrate this point:

1. Topic "Rewriting an algebraic expression without a negative exponent": has a high number of edges to the correct learning state (Node C). We can denote it as a "simple topic" for this example

2. Topic "Determining if graphs have symmetry with respect to the x-axis, y-axis, or origin": has a high number of edges going to the wrong learning state (Node W). We can denote it as a "challenging topic" for this example

We can compute and plot the differences between these two topic networks based on their metrics, as shown in Figure 7.9. Degree centrality measures popularity of the learning states for these topic networks. Reading is more popular for the "easy topic" with respect to the exercises which are more important for the "challenging topic". This might mean, coming back to reading is more important for the "easy topic" with respect to re-attempting exercises being more important for the "challenging topic".

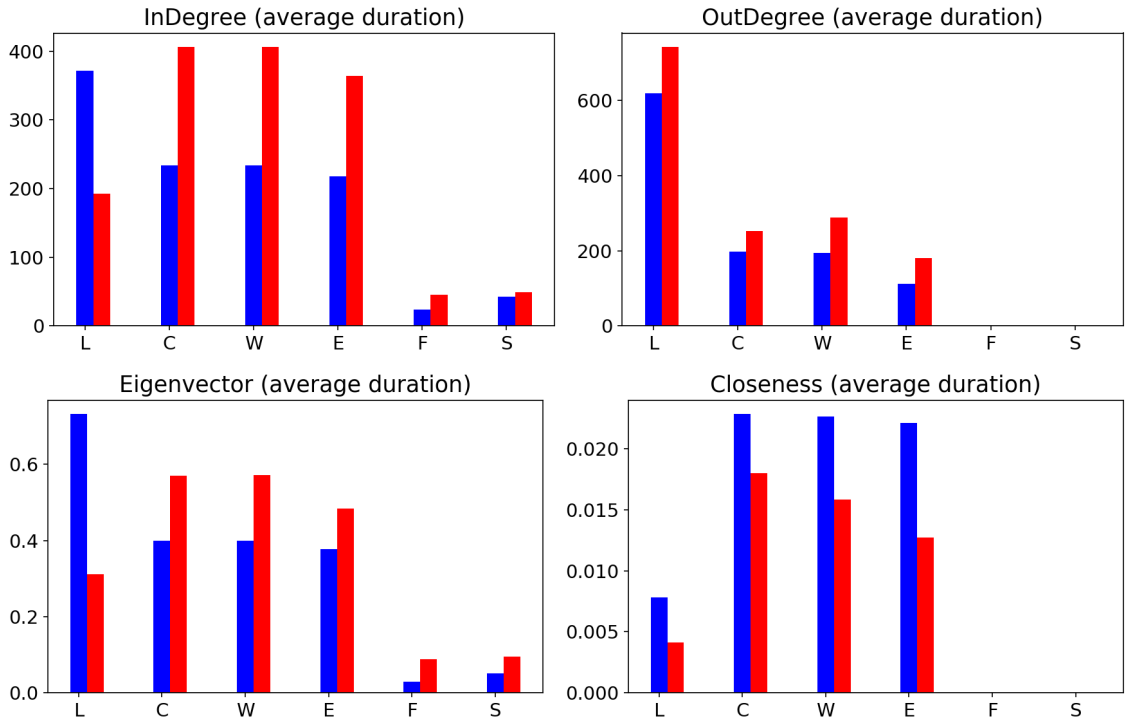


Figure 7.9: Network Degree Metrics Extracted from Two Topic Networks and the Values for Each Learning State. Metrics for One Topic Network are shown in Blue and for the Other Topic Network in Red.

In graph theory, Eigenvector centrality, also called Eigen Centrality, is a measure of the influence of a node in a network [20]. We see similar behaviour as reading is very influential for the "easy topic" with respect to re-attempting exercises being influential for the "challenging topic".

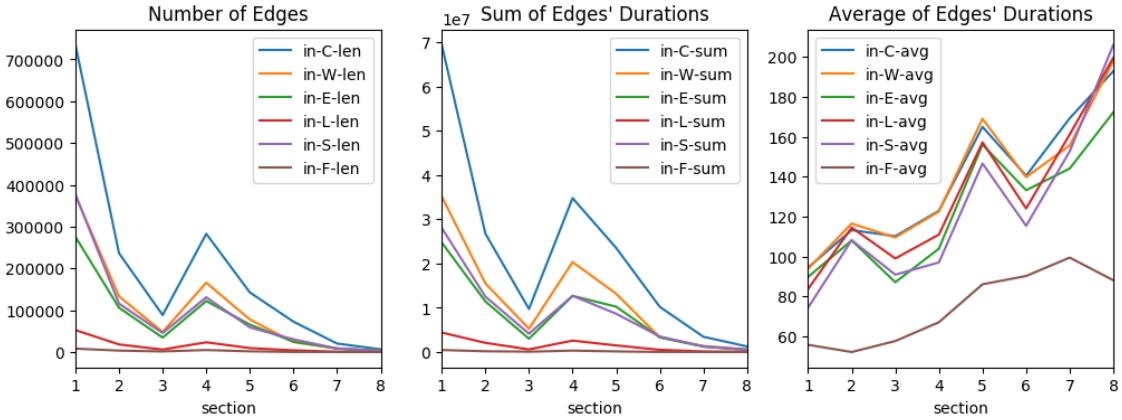
In a connected graph, closeness to centrality of a node is another measure of centrality in a network which is calculated as the reciprocal of the sum of the length of the shortest paths between the node and all other nodes in the graph. Thus, the more central a node is, the closer it is to all other nodes. We can observe the "easy topic" having higher closeness values than the "challenging topic" as the easier topic might have more linear paths and the challenging one having more non-linear paths (as shown in the network examples in Figure 7.8).

In the ASU GFA MOOC, topics belong to sections, also known as slices, which are higher level abstractions of topics. We also develop **directed** networks for each section by using all the transactional data for the topics that belong to each section (or each slice). Figure 7.10 shows metrics for the in-coming and out-going metrics calculated for each of the slices.

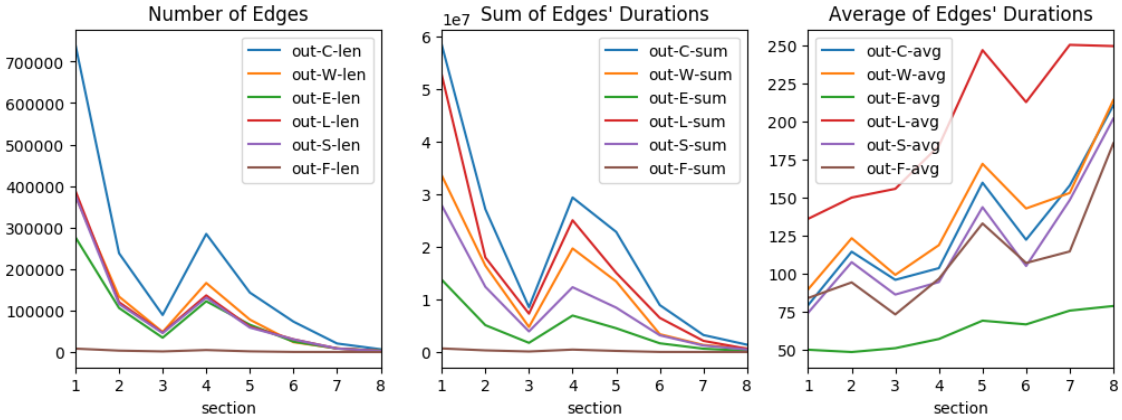
- The first figures, in both Figure 7.10a and Figure 7.10b, show the number of edges when students move from one learning transaction to another, in-coming and out-going respectively. It is clear that the number of students decrease considerably from one section to a more advanced section. From Figure 7.4 shown earlier, we know that only 0.44% of students who started, progressed to Section 8 and only around 5% of students completed 90% of the online content.

- The second figures, for both graphs again, show the sum of all the durations for each of the edges for one transaction learning state to another. Again, in Figure 7.10a we look at the in-coming edges and in Figure 7.10b we look at the out-going edges. The first one shows there are only a few in-coming edges for the Lesson learning state and the second figure illustrates there are only a few out-going edges from the Failed state. We should note that section 4 has a higher number of edges from the transactions, that is because that section contains more concepts than other sections such as section 3, see Figure 7.11 for details.

- The third figures, for both graphs, show the average duration for each of the edges for one transaction learning state to another. These are the most interesting figures. For both in-coming and out-going figures, the more advanced the sections, as measured by graph traversal distances, students will take longer to move to the next learning state, on average. That could mean that either (1) students will take longer on each learning state for more advanced sections, or (2) students who advance further through a course have different learning strategies.



(a) In-coming Network Metrics



(b) Out-going Network Metrics

Figure 7.10: Custom Network Metrics Extracted from the Section Networks Divided by In-coming and Out-going Metrics

The learning states extracted from the transactions and the underlying navigation through the course for which we have presented graph network statistics above, can be modelled using a Markovian procedure [40] by assuming the future learning

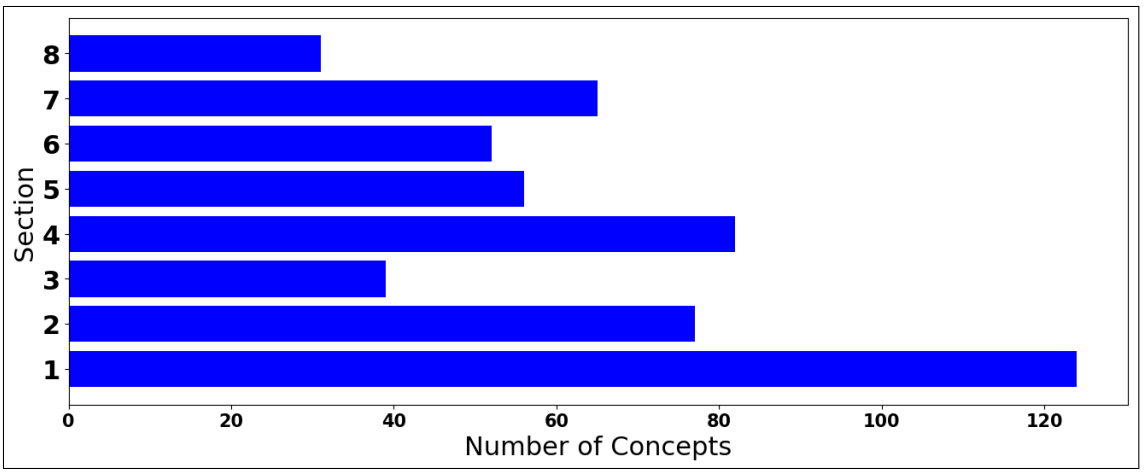


Figure 7.11: Number of Concepts per Section for MAT117 and MAT170 in ASU's GFA via EdX

states depend only on the current learning state. This could be further developed by looking at sequences of learning states and topics learned in order to model the likelihood of future learning states. In our case we consider the learning states as observable states and we can model their learning using an HMM, see Figure 7.12. In this, the unobserved or hidden states can be estimated from the sequence of learning states students follow and are navigated on the system [89].

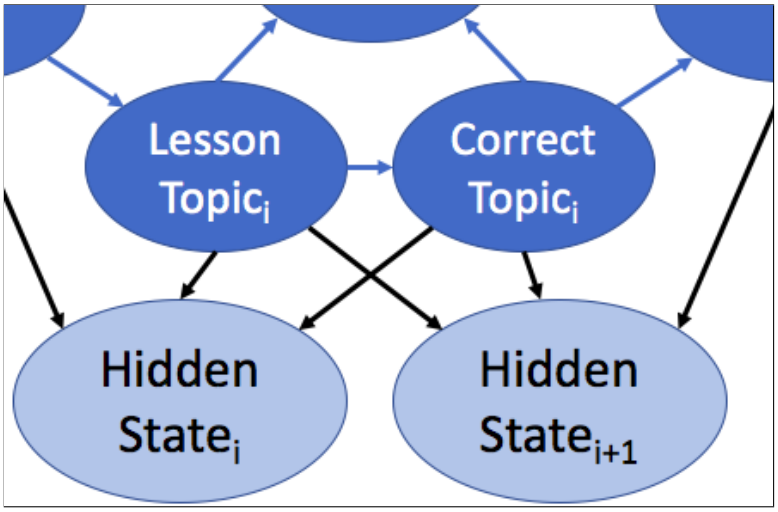


Figure 7.12: Diagram of Modelling How Students Learn on MOOC Platforms using Learning States and Hidden Markov Models

Figure 7.13 shows how the observable states can be mapped to two hidden states.

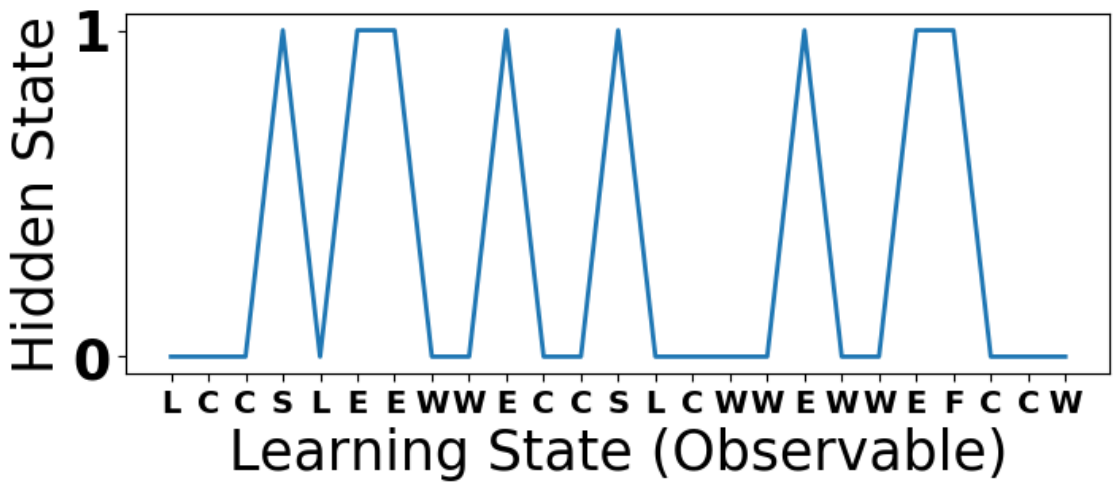


Figure 7.13: Hidden Markov Model trained with Hidden States

7.5 Conclusion on RQ6

Considering the transactional learning states as nodes of a network enables us to analyse their centrality and measure their importance in the learning of these concepts and disciplines. By creating a network for each topic based on all student transactional data, we are measuring the degree centrality of the learning states and discovering where students are most likely to struggle or to succeed by examining the final states and the pathways to those final states. Moreover, we can develop a network with all the learning states and their corresponding topics and analysis the students' progression, searching for centrality hubs, hypothesising whether further interventions may be added, measuring Eigenvector centrality and how important are the neighbours of the nodes, using the geodesic distance to measure the closeness and betweenness between topics for each slice, discovering which slices take more effort from students and how to help them succeed. In addition, we could cluster the nodes and see if the groupings correspond with the sections of the topics.

With this level of analysis possible on log data from learning state transitions on a MOOC, some of which we have actually demonstrated on log files from a MOOC at ASU, we have demonstrated that we can learn useful structure of the transactional learning data using unsupervised approaches, potentially learn valuable insights from

it, and so **RQ6** has been addressed.

Chapter 8

Conclusions

8.1 Introduction

This chapter concludes the thesis by examining each area studied previously, in turn, re-stating some of the conclusions from earlier and enabling some discussion from those outcomes. The central thesis hypothesis was subdivided into smaller areas that were tackled. I have focused on automatically modelling students, automatically detecting students having difficulties in their computer programming modules, automatically offering them assistance and measuring how that aids their learning. Good progress was made towards identifying good predictors of student outcomes and developing interventions for students in computer programming modules.

8.2 Modelling Student Behaviour

Predictive models of student outcome in assessments, using students' characteristics, engagement and programming effort as drivers have proven to contain useful information about their learning progress and understanding. Overall predictions of outcome worked relatively well with only a few years of previous student training data and in the near future we will be able to generalise better across different student cohorts by following a similar approach.

We are eager to add more features to our prediction algorithms which would

enrich the models' representation of students and their activities. These additional features could include the use of the laboratory resources or physical access to buildings in our university. Not only would this better describe our students' digital footprints but it would also allow us to select a subset of the most discriminating features per week or even daily that provides us with an overview of student behaviour to improve the predictive functions early in the semester.

It is important to note we are not tracking every time the student runs a program locally which would be similar to source code snapshots used by other research studies that are being carried out elsewhere. However, we could look at programming states and behaviours between submissions to the platform as additional features.

The techniques developed in the thesis have been deployed in a variety of computer programming modules including CA116, CA117, CA114, CA277 and CA278 at Dublin City University. In general, models were trained using past student data from previous years of the same module and have worked well. However, there were times we used data from a different course or even all data captured in the platform. In those instances, we were not expecting the predictions to work as well as the other models with better training data as the courseware in different modules would not be the same, but they did a good job. The concepts taught are similar but more advanced across different computer programming modules, and utilising patterns from other courses as training dat has surprisingly resulted in good outcomes. The more data there is means classifiers are usually better even if the material and programming exercises are different. This approach could be applicable to other courses not only in computer programming but even Mathematics and other courses with a significant amount of laboratory material or programming work which students need to check and complete on a weekly basis.

8.3 Modelling Students With Embeddings

In this chapter of the thesis, Chapter 5 we explored in some depth, a range of Machine Learning algorithms and techniques including Deep Learning, as we gathered

more student data. Recent work using Deep Learning tends to work better when more and more data is provided. However, in Learning Analytics, the number of students taking a course is an unavoidable limit. Thus we cannot simply generate more data as is done in other domains such as FinTech or Social Network Analysis. Our findings indicate there is a need to learn and to develop better mechanisms to extract and learn effective data features from limited amounts of student data so as to analyse = students' progression and performance effectively.

Our code2vec implementation and results confirm the power of code embeddings as a technique in this application and the latent learning analytics properties in student code submissions. Code embeddings are an alternative to using bag-of-words based representations of source code. In future work we will explore combining Tokens and Abstract Syntax Trees (ASTs) for creating an even richer model of a student that factors in code details from their computer programming assessments including structure and context. In addition, we will explore Concrete Syntax Trees which are parse trees, typically built by a parser during the source code translation and compilation process, adding subsequent processing to ASTs such as contextual information.

User2Code2vec is a novel technique that was developed and used in the thesis to represent students in a high-dimensional space, i.e. when there are many features forming the students' digital footprints. It uses distributional representations of student profiles and their programming code. Other techniques such as Matrix Factorisation can be used to find and to group students with similar coding patterns. In addition, a User Representation Matrix could be built as as a Tensor with a new dimension using all the submissions instead of the last one or one at random. Any of these might give us a better representation of student learning and progression that we could use.

Embeddings have been shown to successfully identify hidden or latent patterns for code submissions and user representations. Measuring the quality of these vectors, however, doing this is not straightforward. Several factors influence the quality

of the vectors such as the amount and quality of the training data, the size of the vectors and the learning algorithm used. The quality of these vectors is crucial for the representations but trying out different hyper-parameters takes a lot of computation and time. Developing and then using pre-trained vectors with a large corpus is a standard approach in other domains such as word vectors developed using Google's News dataset. The Learning Analytics community should make the effort to develop good Code Embeddings that can be used to learn higher level abstracts like User Embeddings.

The use of Embeddings for source code submissions and student code representations is still at an early stage of development but has potential to change how we understand learning to program, recommend code and peer learning of programming using higher level abstractions.

In our future work in this specific area we plan to focus on two main aspects, to learn better distributional semantics using abstract trees to capture syntactic structure effectively following recent work proposed in [83, 6] and to use the recommendation learned using the user2code2vec representation proposed in this work and to evaluate how this representation helps students to improve their learning.

8.4 Providing Adaptive Feedback to Students

In addition to the predicting student outcomes in module assessment, we were also keen to provide a more personalised education to our students by identifying knowledge holes based on their progress in computer programming and the concepts taught in those courses. Such a more personalised education should engage with them more and allow us to automatically redirect them to suitable and appropriate learning material based on their program submissions. This issue of personalised feedback has to be handled with caution as we do not want to redirect students to learning resources where they will not find a solution for a particular submission failure, or to material with which they are already familiar and knowledgable.

Recently, research in Learning Analytics that we have published has focused on

Predictive Modelling and identifying those students having difficulties with course material, also in programming courses [11], and offering remediation, personalised feedback and interventions to students using Machine Learning techniques [9, 10]. Notifying the Lecturers or Professors who deliver computer programming modules and sending personalised assistance to students helped those students at-risk to learn more and reduced the gap in performance in examinations between them and the higher-performing students. It is important to note that higher-performing students do not have the same room for improvement than lower-performing students so for higher-performing students, maintaining their grade is an accomplishment in itself. However, we are trying to measure learning and we expect that lower-performing students tend to learn more in our blended classrooms and complete more assessment programs with mentoring and further assistance.

In terms of the notifications automatically sent to students alerting them about their performance and trajectory, after enabling the feature to opt-in or -out on the module courses, students had to log in to the submission platform to select either option. Most of the students who replied to our survey did sign up and opt-in in the first or second week, specially on the laboratory sessions where they use the platform to verify their work. A few "tardy" students chose to do this opting in later in the module and they did not receive the notifications from weeks they had missed even after they had opted-in. Others were completely disengaged and never got to reply. We should improve the opt-in approach and be more vocal about the project so that every student has the chance to get notifications about their progress.

The approach chosen for the programming recommendations was to pick the closest text program from among those submitted by the top-ranked students in the class that year. This could be further advanced by identifying variables and choosing the closest program syntactically and semantically as we studied in the embeddings chapter, chapter 5. Other approaches tested and deserving of further exploration are Collaborative Filtering as used in recommender systems, by looking at the closest student to a given student, taken from within in the class or from within the top-

students and recommend one of their programs. Netflix recommends movies and Amazon recommends products from people with the same tastes assuming they are constant. It is reasonable to assume that learning computer programming is a stable process and to recommend sample computer programs from those students from their closest person. Last semester, we were more interested in the students being able to identify what is wrong with their programs and the closest solution from a top-student worked very well specially for shorter programs, like the ones suggested in CA114. We also want to explore the use of crowdsourcing and to recommend the program uploaded the most by using previous years solutions if most of the programs remain the same in the future.

In addition, eventually, we would like to go a step further and solve programs for students by suggesting a solution that will fix or improve their own submitted code and that would meet the lecturer's learning criteria. This could be done by tokenising the student's programs to identify the variables used, calculate the similarity and differences with the solutions stored and actually solve the student's program instead of just offering an alternative solution. We need to be cautious with this approach and use it in moderation. However, research shows students also learn by example [92] and a good number of them were demanding more solutions and explanatory guided code in our survey.

8.5 Using Graph Theory and Networks to Model Students

A research visit to Arizona State University (ASU) during the time of this PhD study enabled collaboration in the area of Educational Analytics, a field where both institutions have demonstrated expertise. I was involved with ASU's Action Lab, a dedicated digital teaching and learning laboratory. They are engaging in deep learning analytics, providing continuous program improvement, ultimately resulting in student success. As part of this visit I was able to collect a distinct dataset of

digital footprints from students learning Mathematics at ASU Online via EdX.

To analyse this data, we followed a network analysis approach to investigate the structure of the students' learning on a MOOC system by mapping their learning footprints onto network and graph theory. Network analysis can give us insights into the properties of the learning states including how students progress from one learning state to another. A network of the observed learning states was developed for each student and each topic and posted on a web application.

Some of the findings we encountered in an analysis of this data included the following:

- Students spend the most time working on programs they get correct, than they spend on reading the lessons and then on problems they get incorrect;

- The first slices of learning material and in particular the first that contains the introductory topics is the one that students spent the greatest amount of time on. This is evidence that many students drop out at the beginning of these online courses. However, the topic more students have worked on corresponds to the fifth slice (out of seven) and is called"Adding or subtracting complex numbers". ALEKS technology which is an AI-based recommender system used by students at ASU, redirects students where the students should learn from in the next step of their learning journey, and that concept seems to be very connected to other components in the knowledge space;

- In a similar manner, the two most taken assessment types are the Initial Knowledge Check and the Progress Knowledge Check that are carried out in the beginning and continuously, respectively. However, one of the assignment types taken least by students is the Class Completion Knowledge Check.

What all these findings give us is an insight into student progress, and the high number of drop-outs that are a characteristic of online MOOCs.

8.6 Final thoughts

The main contribution of this thesis, apart from answering a specific set of research questions, is in providing a set of tools that help Lecturers and Professors and encourage students' learning and interest in computer programming by using cutting-edge data mining techniques. I believe computer programming is an ability but also a skill that needs work to help it develop.

As our students demanded in the questionnaire responses they provided, I am eager to provide them with more detailed programming recommendations, suitable material and other actions to fill the knowledge programming holes they may have when learning CS programming in blended classrooms at our university.

Appendices

Appendix A

Publications on Work from this Thesis

A.1 Journal Publications

1. Azcona, D., Hsiao, I.-H., & Smeaton, A. F. (2018). Detecting Students-In-Need in Programming Classes with Multimodal Learning Analytics. Special Issue on Multimodal Learning Analytics & Personalized Support Across Spaces. Journal of User Modeling and User-Adapted Interaction. International Journal of Artificial Intelligence in Education (IjAIED).

2. Azcona, D., & Casey, K. (2015). Micro-analytics for Student Performance Prediction. International Journal of Computer Science and Software Engineering (IJCSSE), 4, 218–223.

A.2 Conference Publications

3. Azcona, D., Arora, P., Hsiao, I.-H., & Smeaton, A. F. (2019). user2code2vec: Embeddings for Profiling Students Based on Distributional Representations of Source Code. In Proceedings of the 9th International Learning Analytics & Knowledge Conference (LAK'19). ACM.

4. Azcona, D., Hsiao, I.-H., & Smeaton, A. F. (2018). PredictCS: Personalizing Programming learning by leveraging learning analytics. In Companion Proceedings of the 8th International Learning Analytics & Knowledge Conference (LAK'18). ACM.

5. Azcona, D., Hsiao, I.-H., & Smeaton, A. F. (2018). Modelling Math Learning on an Open Access Intelligent Tutor. In The 19th International Conference on Artificial Intelligence in Education (AIED 2018).

6. Azcona, D., Hsiao, I.-H., & Smeaton, A. F. (2018). Personalizing Computer Science Education by Leveraging Multimodal Learning Analytics. In Frontiers in Education (FIE 2018).

7. Azcona, D., Hsiao, I.-H., & Smeaton, A. F. (2018). An Exploratory Study on Student Engagement with Adaptive Notifications in Programming Courses. In European Conference on Technology Enhanced Learning (EC-TEL'18). NY, USA: Springer.

8. Vance, Y., Azcona, D., Hsiao, I.-H., & Smeaton, A. F. (2018). Predictive Modelling of Student Reviewing Behaviors in an Introductory Programming Course. In Educational Data Mining in Computer Science Education Workshop (CSEDM'18).

9. Vance, Y., Azcona, D., Hsiao, I.-H., & Smeaton, A. F. (2018). Learning by Reviewing Paper-based Programming Assessments. In European Conference on Technology Enhanced Learning (EC-TEL'18). NY, USA: Springer.

10. Azcona, D., Corrigan, O., Scanlon, P., & Smeaton, A. F. (2017). Innovative learning analytics research at a data-driven HEI. In Third International Conference on Higher Education Advances (HEAd'17). Editorial Universitat Politècnica de València.

11. Azcona, D., & Smeaton, A. F. (2017). Targeting At-risk Students Using Engagement and Effort Predictors in an Introductory Computer Programming

Course. In European Conference on Technology Enhanced Learning (EC-TEL'17) (pp. 361–366). NY, USA: Springer.

A.3 Demos

12. Azcona, D., Moreu, E., Hsiao, I.-H., & Smeaton, A. F. (2019). CoderBot: AI Chatbot to Support Adaptive Feedback for Programming Courses. Companion Proceedings of the 9th International Learning Analytics & Knowledge Conference (LAK'19). ACM.

Appendix B

Organisational Activities

B.1 Workshops

I co-organised the following international workshops:

1. Educational Data Mining in Computer Science Education (CSEDM) Workshop at the University at Buffalo, New York, USA (CSEDM 2018).
   https://sites.google.com/asu.edu/csedm-ws-edm-2018/

2. Educational Data Mining in Computer Science Education (CSEDM) Workshop at Arizona State University, Arizona, USA (CSEDM 2019).
   https://sites.google.com/asu.edu/csedm-ws-lak-2019/

3. Educational Data Mining in Computer Science Education (CSEDM) Workshop at Chicago, USA (CSEDM 2019).
   https://sites.google.com/asu.edu/csedm-ws-aied-2019/

B.2 Proceedings

I was the Proceedings Editor for the 9th International Learning Analytics & Knowledge Conference (LAK 2019) at Arizona State University, Arizona, USA.
https://dl.acm.org/citation.cfm?id=3303772

Appendix C

Presentations on Work from this Thesis

C.1 Selected presentations

1. User2Code2Vec: Embeddings for Profiling Students Based on Distributional Representations of Source Code at Arizona State University, Tempe, AZ, USA (LAK Conference, March 2019)

2. How to win a Hackathon with AI at the Insight Seminar Series (April 2019)

3. Data Mining & Embeddings to Offer Fresh Insights on Irish Politics at the Insight Student Conference 2018 in Unversity College Dublin, Ireland (ISC 2018)

4. Predictive Modelling of Student Reviewing Behaviors in an Introductory Programming Course at the Educational Data Mining in Computer Science Education Workshop at EDM 2018 at the University at Buffalo, New York, USA (CSEDM 2018)

5. PredictCS: Personalizing Programming learning by leveraging learning analytics at the International Workshop on Orchestrating Learning Analytics (OrLA): Learning Analytics Adoption at the Classroom Level in conjunction

with LAK 2018 at The University of Sydney, Australia (LAK 2018)

6. Research & Learn at Arizona State University: Predictions, Recommendations, Embeddings & Networks at Arizona State University's EdPlus in Phoenix, Arizona, USA

7. Targeting At-risk Students Using Engagement and Effort Predictors in an Introductory Computer Programming Course in European Conference on Technology Enhanced Learning in Tallinn University, Estonia (EC-TEL 2017)

8. Innovative learning analytics research at a data-driven HEI at the Third International Conference on Higher Education Advances in Valencia, Spain (HEAd 2017)

9. Presented at the 1st Ireland's National Symposium on Learning Analytics: Bringing People Together, Exploring LA in Irish HE, 2017

10. Crowdsource Programming Recommendations in Educational Analytics at the 3rd Insight Student Conference, Dublin City University, Dublin, Ireland (ISC 2016)

C.2 Other events

11. Demoed at the Learning Analytics & Knowledge 2019 conference in Arizona State University

12. Presented a poster at the 2019 Insight Review in NUI Galway

13. Presented at the Dublin City University's Teaching & Learning Day 2018

14. Presented a poster at the International Conference on Artificial Intelligence in Education (AIED 2018)

15. Demoed about Educational Analytics in Computer Science at the 1st. Insight Augmented Human Demonstrator Event in March 2017

16. Attended the Fulbright Enrichment April 2018 Seminar Philadelphia, Pennsylvania, USA

17. Attended the Amazon's AWS re:Invent 2017 in Las Vegas, Nevada, USA

18. Presented a poster at the Ireland's Data Summit where I met Leo Varadkar the Taoiseach (Prime Minister) in Ireland, 2017

19. Attended the Big Data and Analytics Summer School at University of Essex in September 2016 as a YERUN awardee

20. Presented posters at Dublin City University's Faculty of Engineering and Computing Research Day in 2017 and 2018

Appendix D

Awards

- Government of Ireland Doctoral Awardee (2015-2019) by the Irish Research Council in partnership with the National Forum for the Enhancement of Teaching and Learning in Ireland (2015-2019)
- Fulbright Visiting Researcher at Arizona State University, USA during the 2017/2018 academic year
- Winner of the Ulster Bank Hackathon (2019) at Dogpatch Labs Dublin, Ireland
- Winner of the Geek Challenge (2018) by Novicell at Denmark
- Special Mention at StartUp Weekend Dublin (2018) by Bank of Ireland in Google, Dublin, Ireland
- Huawei's Future of Vision Challenge Awardee (2018) at Trinity College Dublin, Ireland
- Microsoft Imagine Cup Awardee (2018) in San Francisco and Seattle, USA. National Awards: Best Use of Artificial Intelligence & 4th place at USA Nationals. World Awards: Top-6 in Artificial Intelligence & Semifinalists at World Finals
- Grant for a coding non-profit at Arizona State University (2018) by ASU Changemaker

- Invited scholar at Data Science Conference (2017) by Open Data Science Conference in San Francisco, CA, USA
- Invited scholar at Analytics Summer School (2017) by Young European Research Universities Network (YERUN) in University of Essex, UK
- Postgraduate Accommodation scholar by Faculty of Engineering and Computing, DCU, (2015-2016)

Bibliography

[1] Alireza Ahadi et al. “Exploring machine learning methods to automatically identify students in need of assistance”. In: *Proceedings of the Eleventh Annual International Conference on International Computing Education Research*. ACM. 2015, pp. 121–130.

[2] Davide Albanese et al. “Minerva and minepy: a C engine for the MINE suite and its R, Python and MATLAB wrappers”. In: *Bioinformatics* 29.3 (2012), pp. 407–408.

[3] Miltiadis Allamanis, Hao Peng, and Charles Sutton. “A convolutional attention network for extreme summarization of source code”. In: *International Conference on Machine Learning*. 2016, pp. 2091–2100.

[4] Miltiadis Allamanis et al. “Learning natural coding conventions”. In: *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*. ACM. 2014, pp. 281–293.

[5] Uri Alon et al. “A general path-based representation for predicting program properties”. In: *arXiv preprint arXiv:1803.09544* (2018).

[6] Uri Alon et al. “code2vec: Learning Distributed Representations of Code”. In: *arXiv preprint arXiv:1803.09473* (2018).

[7] Kimberly E Arnold and Matthew D Pistilli. “Course Signals at Purdue: Using learning analytics to increase student success”. In: *Proceedings of the 2nd International Conference on Learning Analytics and Knowledge (LAK)*. ACM. 2012, pp. 267–270.

[8] David Azcona and Kevin Casey. "Micro-analytics for Student Performance Prediction". In: *International Journal of Computer Science and Software Engineering (IJCSSE)* 4.8 (2015), pp. 218–223.

[9] David Azcona, I-Han Hsiao, and Alan F Smeaton. "Detecting Students-In-Need in Programming Classes with Multimodal Learning Analytics". In: *International Journal of Artificial Intelligence in Education (ijAIED)* (2018).

[10] David Azcona, I-Han Hsiao, and Alan F Smeaton. "PredictCS: Personalizing Programming Learning by Leveraging Learning Analytics". In: *Companion Proceedings 8th International Conference on Learning Analytics & Knowledge (LAK)* (2018).

[11] David Azcona and Alan F Smeaton. "Targeting At-risk Students Using Engagement and Effort Predictors in an Introductory Computer Programming Course". In: *European Conference on Technology Enhanced Learning*. Springer. 2017, pp. 361–366.

[12] Lei Bao. "Theoretical comparisons of average normalized gain calculations". In: *American Journal of Physics* 74.10 (2006), pp. 917–922.

[13] Marco Baroni, Georgiana Dinu, and Germán Kruszewski. "Don't count, predict! A systematic comparison of context-counting vs. context-predicting semantic vectors". In: *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Vol. 1. 2014, pp. 238–247.

[14] Yoshua Bengio et al. "A neural probabilistic language model". In: *Journal of Machine Learning Research* 3.Feb (2003), pp. 1137–1155.

[15] Jens Bennedsen and Michael E Caspersen. "Failure rates in introductory programming". In: *ACM SIGCSE Bulletin* 39.2 (2007), pp. 32–36.

[16] Susan Bergin and Ronan Reilly. "Predicting introductory programming performance: A multi-institutional multivariate study". In: *Computer Science Education* 16.4 (2006), pp. 303–323.

[17] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

[18] Paulo Blikstein. "Using learning analytics to assess students' behavior in open-ended programming tasks". In: *Proceedings of the 1st International Conference on Learning Analytics and Knowledge (LAK)*. ACM. 2011, pp. 110–116.

[19] Paulo Blikstein and Marcelo Worsley. "Multimodal Learning Analytics and Education Data Mining: using computational technologies to measure complex learning tasks". In: *Journal of Learning Analytics* 3.2 (2016), pp. 220–238.

[20] Phillip Bonacich. "Some unique properties of eigenvector centrality". In: *Social networks* 29.4 (2007), pp. 555–564.

[21] Kristy Elizabeth Boyer et al. "Balancing cognitive and motivational scaffolding in tutorial dialogue". In: *International Conference on Intelligent Tutoring Systems*. Springer. 2008, pp. 239–249.

[22] Neil Christopher Charles Brown et al. "Blackbox: a large scale repository of novice programmers' activity". In: *Proceedings of the 45th ACM technical symposium on Computer science education*. ACM. 2014, pp. 223–228.

[23] Adam S Carter, Christopher D Hundhausen, and Olusola Adesope. "The normalized programming state model: Predicting student performance in computing courses based on programming behavior". In: *Proceedings of the Eleventh Annual International Conference on International Computing Education Research*. ACM. 2015, pp. 141–150.

[24] Adam Scott Carter and Christopher David Hundhausen. "Using Programming Process Data to Detect Differences in Students' Patterns of Programming". In: *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*. ACM. 2017, pp. 105–110.

[25] Ana Carvalho, Nelson Areal, and Joaquim Silva. "Students' perceptions of Blackboard and Moodle in a Portuguese university". In: *British Journal of Educational Technology* 42.5 (2011), pp. 824–841.

[26] Weiyu Chen et al. "Early Detection Prediction of Learning Outcomes in Online Short-Courses via Learning Behaviors". In: *IEEE Transactions on Learning Technologies* 12 (1 2019), pp. 44–58.

[27] Marc Claesen and Bart De Moor. "Hyperparameter search in machine learning". In: *arXiv preprint arXiv:1502.02127* (2015).

[28] Owen Corrigan et al. "Using Educational Analytics to Improve Test Performance". In: *Design for Teaching and Learning in a Networked World*. Springer, 2015, pp. 42–55.

[29] Scotty D Craig et al. "Learning with ALEKS: The Impact of Students' Attendance in a Mathematics After-School Program". In: *International Conference on Artificial Intelligence in Education*. Springer. 2011, pp. 435–437.

[30] Scotty D Craig et al. "The impact of a technology-based mathematics after-school program using ALEKS on student's knowledge and behaviors". In: *Computers & Education* 68 (2013), pp. 495–504.

[31] Maria Cutumisu and Daniel L Schwartz. "Choosing versus Receiving Feedback: The Impact of Feedback Valence on Learning in an Assessment Game." In: *Proceedings of the 9th International Conference on Educational Data Mining (EDM)*. 2016, pp. 341–346.

[32] Matt Dennis, Judith Masthoff, and Chris Mellish. "Adapting progress feedback and emotional support to learner personality". In: *International Journal of Artificial Intelligence in Education* 26.3 (2016), pp. 877–931.

[33] Luc Devroye, László Györfi, and Gábor Lugosi. *A probabilistic theory of pattern recognition*. Vol. 31. Springer Science & Business Media, 2013.

[34] Nicholas Diana et al. “An instructor dashboard for real-time analytics in interactive programming assignments”. In: *Proceedings of the Seventh International Learning Analytics & Knowledge Conference (LAK)*. ACM. 2017, pp. 272–279.

[35] Jean-Paul Doignon and Jean-Claude Falmagne. “Spaces for the assessment of knowledge”. In: *International journal of man-machine studies* 23.2 (1985), pp. 175–196.

[36] Jean-Paul Doignon, Jean-Claude Falmagne, and Eric Cosyn. “Learning Spaces: A Mathematical Compendium”. In: *Knowledge Spaces*. Springer, 2013, pp. 131–145.

[37] Stephen H Edwards and Krishnan Panamalai Murali. “CodeWorkout: short programming exercises with built-in data collection”. In: *Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education*. ACM. 2017, pp. 188–193.

[38] Stephen H Edwards and Manuel A Perez-Quinones. “Web-CAT: automatically grading programming assignments”. In: *ACM SIGCSE Bulletin*. Vol. 40. 3. ACM. 2008, pp. 328–328.

[39] Anthony Estey, Hieke Keuning, and Yvonne Coady. “Automatically Classifying Students in Need of Support by Detecting Changes in Programming Behaviour”. In: *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*. ACM. 2017, pp. 189–194.

[40] Paul A Gagniuc. *Markov chains: from theory to implementation and experimentation*. John Wiley & Sons, 2017.

[41] Pierre Geurts, Damien Ernst, and Louis Wehenkel. “Extremely randomized trees”. In: *Machine learning* 63.1 (2006), pp. 3–42.

[42] Ross Girshick et al. “Rich feature hierarchies for accurate object detection and semantic segmentation”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2014, pp. 580–587.

[43] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. “Deep learning”. In: (2016).

[44] Sebastian Gross et al. “Example-based feedback provision using structured solution spaces”. In: *International Journal of Learning Technology 10* 9.3 (2014), pp. 248–280.

[45] L Gyorfi, L Devroye, and G Lugosi. *A probabilistic theory of pattern recognition*. Washington, USA, 1996.

[46] Bjorn Hartmann et al. “What would other programmers do: suggesting solutions to error messages”. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM. 2010, pp. 1019–1028.

[47] John Hattie and Helen Timperley. “The power of feedback”. In: *Review of Educational Research* 77.1 (2007), pp. 81–112.

[48] Arto Hellas et al. “Predicting academic performance: a systematic literature review”. In: *Proceedings Companion of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education*. ACM. 2018, pp. 175–199.

[49] I-H Hsiao, Sergey Sosnovsky, and Peter Brusilovsky. “Guiding students to the right questions: adaptive navigation support in an E-Learning system for Java programming”. In: *Journal of Computer Assisted Learning* 26.4 (2010), pp. 270–283.

[50] I-Han Hsiao, Po-Kai Huang, and Hannah Murphy. “Uncovering reviewing and reflecting behaviors from paper-based formal assessment”. In: *Proceedings of the Seventh International Learning Analytics & Knowledge Conference*. ACM. 2017, pp. 319–328.

[51] Petri Ihantola et al. “Educational data mining and learning analytics in programming: Literature review and case studies”. In: *Proceedings of the 2015 ITiCSE on Working Group Reports*. ACM. 2015, pp. 41–63.

[52] Petri Ihantola et al. “Review of recent systems for automatic assessment of programming assignments”. In: *Proceedings of the 10th Koli calling international conference on computing education research*. ACM. 2010, pp. 86–93.

[53] G Tanner Jackson and Arthur C Graesser. “Content matters: An investigation of feedback categories within an ITS”. In: *Frontiers in Artificial Intelligence and Applications* 158 (2007), p. 127.

[54] Matthew C Jadud. “Methods and tools for exploring novice compilation behaviour”. In: *Proceedings of the Second International Workshop on Computing Education Research*. ACM. 2006, pp. 73–84.

[55] George F Jenks. “The data model concept in statistical mapping”. In: *International yearbook of cartography* 7 (1967), pp. 186–190.

[56] Ian Jolliffe. *Principal component analysis*. Springer, 2011.

[57] John D Kelleher, Brian Mac Namee, and Aoife D’arcy. *Fundamentals of machine learning for predictive data analytics: algorithms, worked examples, and case studies*. MIT Press, 2015.

[58] Hassan Khosravi and Kendra ML Cooper. “Using Learning Analytics to Investigate Patterns of Performance and Engagement in Large Classes”. In: *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*. ACM. 2017, pp. 309–314.

[59] Michael Kölling et al. “The BlueJ system and its pedagogy”. In: *Computer Science Education* 13.4 (2003), pp. 249–268.

[60] Jakub Kuzilek et al. “OU analyse: Analysing at-risk students at the Open University”. In: *Learning Analytics Review* (2015), pp. 1–16.

[61] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. “Deep learning”. In: *nature* 521.7553 (2015), p. 436.

[62] Wenting Ma et al. “Intelligent tutoring systems and learning outcomes: A meta-analysis.” In: *Journal of Educational Psychology* 106.4 (2014), pp. 901–918.

[63] Christopher D Manning, Prabhakar Raghavan, and Hinrich Schütze. "Text classification and naive Bayes". In: *Introduction to information retrieval* 1.6 (2008).

[64] Nikos Manouselis et al. "Recommender systems in technology enhanced learning". In: *Recommender Systems Handbook*. Springer, 2011, pp. 387–415.

[65] Richard Meyes et al. "Ablation Studies in Artificial Neural Networks". In: *arXiv preprint arXiv:1901.08644* (2019).

[66] Tomas Mikolov et al. "Distributed representations of words and phrases and their compositionality". In: *Advances in neural information processing systems*. 2013, pp. 3111–3119.

[67] Tomas Mikolov et al. "Efficient estimation of word representations in vector space". In: *arXiv preprint arXiv:1301.3781* (2013).

[68] Tripti Mishra, Dharminder Kumar, and Sangeeta Gupta. "Mining students' data for prediction performance". In: *2014 Fourth International Conference on Advanced Computing & Communication Technologies*. IEEE. 2014, pp. 255–262.

[69] Ewa Młynarska, Derek Greene, and Pádraig Cunningham. "Indicators of good student performance in moodle activity data". In: *arXiv preprint arXiv:1601.02975* (2016).

[70] Oliver Mooney et al. "A study of progression in Irish higher education". In: *Dublin: Higher Education Authority* (2010).

[71] Lili Mou et al. "Building program vector representations for deep learning". In: *arXiv preprint arXiv:1409.3358* (2014).

[72] Lili Mou et al. "Convolutional Neural Networks over Tree Structures for Programming Language Processing." In: *AAAI*. Vol. 2. 3. 2016, p. 4.

[73] Dhawal Mujumdar et al. "Crowdsourcing suggestions to programming problems for dynamic web development languages". In: *CHI'11 Extended Abstracts on Human Factors in Computing Systems*. ACM. 2011, pp. 1525–1530.

[74] Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

[75] Susanne Narciss. "Feedback strategies for interactive learning tasks". In: *Handbook of research on educational communications and technology* 3 (2008), pp. 125–144.

[76] Susanne Narciss et al. "Exploring feedback and student characteristics relevant for personalizing feedback strategies". In: *Computers & Education* 71 (2014), pp. 56–76.

[77] Iulian Neamtiu, Jeffrey S Foster, and Michael Hicks. "Understanding source code evolution using abstract syntax tree matching". In: *ACM SIGSOFT Software Engineering Notes* 30.4 (2005), pp. 1–5.

[78] Andrew Ng. "What data scientists should know about deep learning". In: . 44 (2015).

[79] Xabier Ochoa. "Multimodal Learning Analytics". In: *The Handbook of Learning Analytics, 1 ed., C. Lang, G. Siemens, A. F. Wise and D. Gasevic, Eds.* Alberta, Canada: Society for Learning Analytics Research (SoLAR), 2017, pp. 129–141.

[80] Benjamin Paaßen et al. "The Continuous Hint Factory-Providing Hints in Vast and Sparsely Populated Edit Distance Spaces". In: *arXiv preprint arXiv:1708.06564* (2017).

[81] Andrei Papancea, Jaime Spacco, and David Hovemeyer. "An open platform for managing short programming exercises". In: *Proceedings of the ninth annual international ACM conference on International computing education research*. ACM. 2013, pp. 47–52.

[82] Andrew Petersen et al. “Revisiting why students drop CS1”. In: *Proceedings of the 16th Koli Calling International Conference on Computing Education Research*. ACM. 2016, pp. 71–80.

[83] Chris Piech et al. “Learning program embeddings to propagate feedback on student code”. In: *Proceedings of the 32nd International Conference on International Conference on Machine Learning-Volume 37*. JMLR. org. 2015, pp. 1093–1102.

[84] Chris Piech et al. “Modeling how students learn to program”. In: *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education*. ACM. 2012, pp. 153–160.

[85] Victor Pigott and Denise Frawley. “An Analysis of Completion in Irish Higher Education: 2007/08 Entrants”. In: *Higher Education Authority in Ireland* (2019).

[86] Sebastian Proksch, Sven Amann, and Sarah Nadi. “Enriched event streams: a general dataset for empirical studies on in-IDE activities of software developers”. In: *Proceedings of the International Conference on Mining Software Repositories*. 2018.

[87] Sebastian Proksch et al. “A dataset of simplified syntax trees for C”. In: *Proceedings of the 13th International Conference on Mining Software Repositories*. ACM. 2016, pp. 476–479.

[88] Keith Quille, Susan Bergin, and Aidan Mooney. “PreSS#, A Web-Based Educational System to Predict Programming Performance”. In: *International Journal of Computer Science and Software Engineering (IJCSSE)* 4.7 (2015), pp. 178–189.

[89] Lawrence R Rabiner. “A tutorial on hidden Markov models and selected applications in speech recognition”. In: *Proceedings of the IEEE* 77.2 (1989), pp. 257–286.

[90] Maxim Rabinovich, Mitchell Stern, and Dan Klein. "Abstract syntax networks for code generation and semantic parsing". In: *arXiv preprint arXiv:1704.07535* (2017).

[91] Veselin Raychev, Martin Vechev, and Eran Yahav. "Code completion with statistical language models". In: *ACM Sigplan Notices*. Vol. 49. 6. ACM. 2014, pp. 419–428.

[92] Rebecca Reynolds and Ming Ming Chiu. "Formal and informal context factors as contributors to student engagement in a guided discovery-based program of game design learning". In: *Learning, Media and Technology* 38.4 (2013), pp. 429–462.

[93] Shaghayegh Sahebi, Yu-Ru Lin, and Peter Brusilovsky. "Tensor factorization for student modeling and performance prediction in unstructured domain". In: *Proceedings of the 9th International Conference on Educational Data Mining*. IEDMS. 2016, pp. 502–506.

[94] Gerard Salton, Anita Wong, and Chung-Shu Yang. "A vector space model for automatic indexing". In: *Communications of the ACM* 18.11 (1975), pp. 613–620.

[95] Valerie J Shute and Diego Zapata-Rivera. "Adaptive technologies". In: *ETS Research Report Series* 2007.1 (2007).

[96] George Siemens and Phil Long. "Penetrating the fog: Analytics in learning and education." In: *EDUCAUSE review* 46.5 (2011), p. 30.

[97] Shashank Singh. "CodeReco-A Semantic Java Method Recommender". PhD thesis. Arizona State University, 2017.

[98] Sergey Sosnovsky, I-Han Hsiao, and Peter Brusilovsky. "Adaptation "in the Wild": ontology-based personalization of open-corpus learning material". In: *European Conference on Technology Enhanced Learning*. Springer. 2012, pp. 425–431.

[99] Terry Speed. “A correlation for the 21st century”. In: *Science* 334.6062 (2011), pp. 1502–1503.

[100] Sergios Theodoridis, Konstantinos Koutroumbas, et al. “Pattern recognition”. In: *IEEE Transactions on Neural Networks* 19.2 (2008), p. 376.

[101] Michael E Tipping and Christopher M Bishop. “Probabilistic principal component analysis”. In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 61.3 (1999), pp. 611–622.

[102] Christopher Watson and Frederick WB Li. “Failure rates in introductory programming revisited”. In: *Proc. 2014 Conference on Innovation & Technology in Computer Science Education*. ACM. 2014, pp. 39–44.

[103] Christopher Watson, Frederick WB Li, and Jamie L Godwin. “Bluefix: Using crowd-sourced feedback to support programming students in error diagnosis and repair”. In: *International Conference on Web-Based Learning*. Springer. 2012, pp. 228–239.

[104] Christopher Watson, Frederick WB Li, and Jamie L Godwin. “No tests required: comparing traditional and dynamic predictors of programming success”. In: *Proceedings of the 45th ACM Technical Symposium on Computer Science Education*. ACM. 2014, pp. 469–474.

[105] Christopher Watson, Frederick WB Li, and Jamie L Godwin. “Predicting performance in an introductory programming course by logging and analyzing student programming behavior”. In: *Advanced Learning Technologies (ICALT), 2013 IEEE 13th International Conference on*. IEEE. 2013, pp. 319–323.

[106] Bernard L Welch. “The generalization ofstudent’s’ problem when several different population variances are involved”. In: *Biometrika* 34.1/2 (1947), pp. 28–35.

[107] Doug Wightman et al. “Snipmatch: using source code context to enhance snippet retrieval and parameterization”. In: *Proceedings of the 25th Annual ACM Symposium on User Interface Software and Technology*. ACM. 2012, pp. 219–228.

[108] Ben Williamson. “The hidden architecture of higher education: building a big data infrastructure for the ‘smarter university’”. In: *International Journal of Educational Technology in Higher Education* 15.1 (2018), p. 12.

[109] Annika Wolff et al. “Improving retention: predicting at-risk students by analysing clicking behaviour in a virtual learning environment”. In: *Proceedings of the third international conference on learning analytics and knowledge (LAK)*. ACM. 2013, pp. 145–149.

[110] Billy Tak-ming Wong and Kam Cheong Li. “A review of learning analytics intervention in higher education (2011–2018)”. In: *Journal of Computers in Education* (2019), pp. 1–22.

[111] Yunwen Ye and Gerhard Fischer. “Reuse-conducive development environments”. In: *Automated Software Engineering* 12.2 (2005), pp. 199–235.